
Algorithm Toolkit Documentation

Release 0.1.0

Chris Willey, Brent Bartlett, Jason Casey

Feb 16, 2021

Contents

1	Contents	3
1.1	About the Algorithm Toolkit	3
1.2	Installation	4
1.3	PyCharm Configuration for ATK	7
1.4	Algorithm Projects	9
1.5	The ATK Development Environment	12
1.6	Working With Algorithms	17
1.7	Creating Your First Algorithm	28
1.8	Working With Chains	41
1.9	The Chain Ledger	48
1.10	Tutorial: Do Maths	51
1.11	CLI	66
1.12	API	70
1.13	Docker	73
1.14	The Algorithm Registry	75
1.15	Contributing to the ATK	78
1.16	Indices and tables	80
	Index	81

This project provides an easy way for researchers and developers to develop and share algorithms related to geospatial data and imagery.

The source code for this project lives on [GitHub](#).

1.1 About the Algorithm Toolkit

1.1.1 Why does this exist?

Algorithms on this site can benefit people and industries. From helping investors predict markets more accurately to making farmers more productive, image processing algorithms are revolutionizing the way we live and work. For too long these have been locked away in academia or in private companies. We thought it was time to bring them to the world.

1.1.2 What are algorithms?

Images are data. For each pixel in an image you may have hundreds or even thousands of data values captured by sophisticated sensor equipment, or attached by software during image processing. Algorithms are basically mathematical equations applied to the data contained in images in order to reveal aspects of the image that may not normally be visible.

1.1.3 What is the ATK?

Researchers all over the world have written software programs that apply algorithms to images. Up to now, those programs have been written in lots of different ways using lots of different software tools, and haven't been able to talk to other programs written by other researchers. Our ATK creates a way for anyone who develops one of these software programs to make it available in a standard way.

1.1.4 Doesn't sound like such a big deal.

Typically, many algorithms are used in sequence to process an image. For example, if you want to show vegetation health from pictures taken from a satellite, you might:

- Download the pictures from the satellite's feed

- Stitch together the pictures into a single orthomosaic image
- Perform a calculation called NDVI to show amounts of chlorophyll in each pixel of that image
- Apply a color scale to the resulting pixels (which otherwise would be grayscale); maybe green to red
- Export the image to the web

Each of these steps might be done by a single algorithm, but how does each step know what to do with the stuff that came before it? And how does it know what to provide the next step?

The ATK creates a way to define an algorithm “chain” with standard inputs and outputs. That way, when you create an algorithm you know what you’ll be getting in and can tell other developers what you’re providing out.

1.2 Installation

1.2.1 Super Quick Start

Python projects should be installed in virtual environments in order to keep versions of various packages in sync with the project code. However, if you want to get up and running immediately you can do the following in a Terminal window (assumes Python and pip are installed):

```
pip install algorithm_toolkit
alg cp myproject
cd myproject
alg run
```

Point your browser to <http://localhost:5000/>. You should see the development environment welcome page.

See the next section for a more detailed install process.

1.2.2 Slightly More Involved But Still Pretty Quick Start

1.2.3 A word about virtual environments

As noted above, it’s a best practice to start new Python projects in virtual environments. As you work on code, you rely more and more on external libraries. As time passes, changes to those libraries will eventually break your code unless you are constantly updating it. So when you install a new version of a library for another project, suddenly you find that your earlier project no longer works.

Virtual environments solve this problem. Each environment contains a discreet set of the libraries used by your code, at the versions you determine.

Thankfully, creating virtual environments in Python is easy.

Python 3

Linux and Mac:

```
python3 -m venv myenvironment
source myenvironment/bin/activate
```

Note: On some Linux systems, python3-venv is not installed by default. If you get an error with the command above, try:


```
# Debian/Ubuntu
sudo apt install python3-venv

# RedHat/CentOS
sudo yum install python3-venv
```

On Windows:

```
py -3 -m venv myenvironment
myenvironment\Scripts\activate.bat
```

Python 2

Python 2 does not come with a built in virtual environment creator. We recommend using `virtualenvwrapper`. [Their install docs are excellent](#).

Once you’ve installed and configured `virtualenvwrapper`, create your virtual environment:

```
mkvirtualenv myenvironment
```

1.2.4 Setting up your ATK project

```
pip install algorithm_toolkit
alg cp myproject
cd myproject
alg run
```

Point your browser to <http://localhost:5000/>. You should see the development environment welcome page.

What just happened?

Let’s walk through this line by line.

```
pip install algorithm_toolkit
```

The ATK lives on `PyPi`, so this line downloads and installs the ATK in your virtual environment. Several dependencies will be installed as well.

```
alg cp myproject
```

This line uses the ATK’s Command Line Interface (CLI) called `alg`. The `cp` command stands for “create project”. “myproject” is the name of your project, which will also be the name of the folder created for the project (feel free to use a more original name).

```
cd myproject
```

Puts you in the project folder.

```
alg run
```

This command also uses the CLI. The `run` command starts a development web server. As we discuss elsewhere in the docs (see TODO: create page), setting up your algorithms and processing chains is accomplished using a web-based interface.

1.2.5 Installing the example project

The ATK comes with an example project to help you understand how it works.

Prerequisites

The example project has some additional dependencies, including [NumPy](#), in order to work. If you're on a Linux machine, you can install the example project and it will handle this dependency for you.

However, if you're on a Mac or Windows machine, installing NumPy is more complicated.

Anaconda

For these operating systems, we highly recommend using Anaconda or its smaller cousin Miniconda. The only difference between these two is that Anaconda installs over 150 packages (including SciPy and NumPy) out of the box whereas with Miniconda you need to install everything separately. Either way is fine.

- [Anaconda installation Page](#)
- [Miniconda installation Page](#)

Once Anaconda or Miniconda is installed do the following:

```
conda install Pillow requests numpy Shapely
```

Install the example project

To set up the example project, just use the `--example` flag when setting up a new project:

```
alg cp myproject --example
```

1.2.6 Installing documentation locally

If you want these docs to be installed locally, use the `--with-docs` flag when creating a project. You need to have Sphinx installed for this to work.

```
pip install Sphinx sphinx_rtd_theme
alg cp myproject --with-docs
```

1.2.7 Troubleshooting Install Issues

On some systems, additional libraries may be needed to install the Algorithm Toolkit. Try these packages if your ATK install fails:

Debian/Ubuntu Linux

```
# python 3
sudo apt install python3-dev build-essential

# python 2
sudo apt install python-dev build-essential
```

RedHat/CentOS Linux

```
# python 3
sudo yum install python3-devel

# python 2
sudo yum install python-devel
```

1.3 PyCharm Configuration for ATK

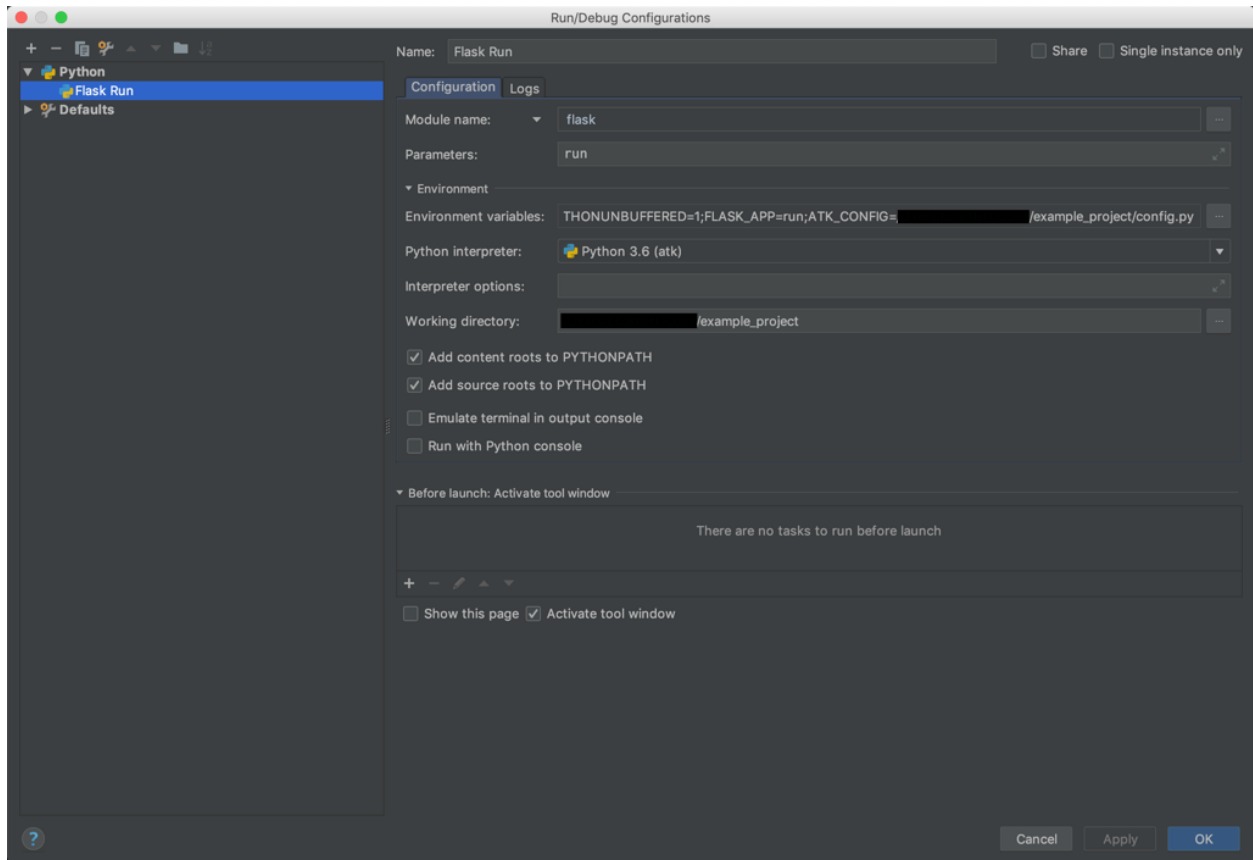
If you use [PyCharm](#) to develop Python applications, you can set it up to run the development environment for you rather than using the command line. The benefit of this is that you can then use the PyCharm debugging tools to set breakpoints and find out what's going on with your application.

This page will explain how to set up PyCharm correctly. It assumes you know how to use the PyCharm debugger.

1.3.1 Adding a Configuration

Before you set up PyCharm, create your virtual environment and first project as described in [Installation](#).

Once you've done this, in PyCharm select Edit Configurations from the Run menu. You'll see a screen like this:



Click the “+” sign to Add New Configuration.

In the screenshot we’ve filled in the fields of information you’ll need:

- **Name:** Give this configuration a name (like “Run ATK server”).
- **Single instance only:** check this box.
- **Module name:** when you first open this screen, the first field will read “Script path”. Click the small arrow next to the name to reveal the option “Module name”. Type the word `flask` into this field.

Note: As we mention elsewhere in the docs, each Algorithm Project is a [Flask](#) app.

- **Parameters:** type the word `run` into this field.
- **Environment variables:** there will already be a variable called `PYTHONBUFFERED=1` in this field. Add a semicolon after that, and add: `FLASK_APP=run;ATK_CONFIG=` plus the full path to the `config.py` file in your Algorithm Project (e.g.: `/home/myuser/atk/myproject/config.py`). The whole line will read: `PYTHONBUFFERED=1;FLASK_APP=run;ATK_CONFIG=/home/myuser/atk/myproject/config.py`.

Note: On Windows, the path to the `config.py` file would be indicated more like this: `C:/Users/myuser/atk/myproject/config.py`. Note the use of forward slashes instead of the more typical back slashes.

- **Python interpreter:** the version of Python *inside your virtual environment*. This is important, because the Python interpreter is linked to the environment containing all the project dependencies (like the Algorithm

Toolkit itself). If you use a different interpreter PyCharm will not find the Python packages it needs.

Note: If you don't see the interpreter for your virtual environment, select "Add Local" from the interpreter drop-down list. Then find the path to the python executable. On Mac OS and Linux, you can find it using `which python`. On Windows you may be able to use `where python` in the same way. NOTE that you must be in the virtual environment for this to work.

- **Working directory:** the full path to your example project's root folder.

Note: Unlike for the Environment variables field, on Windows this path will have back slashes instead of forward slashes.

1.3.2 Running the server

Once you've saved this configuration, you can run the server using the Play button next to the new configuration.

1.4 Algorithm Projects

When you use the Algorithm Toolkit (ATK), you will create and work on an Algorithm Project. A project contains one or more algorithms that you write, or that you install from the Algorithm Registry. It also contains some code to configure and run the project itself.

Creating a new project using the CLI will also create all the files and folders you need to get started. A new, empty project will look like this:

```
algorithms/  
  __init__.py  
chains/  
logs/  
  app.log  
.env  
.gitignore  
__init__.py  
config.py  
licenses.json  
run.py
```

Let's look at what each of these files and folders do:

- The **algorithms/** folder will contain the algorithms used by this project. As you create or install algorithms, they will be placed in this folder.
- The **chains/** folder will contain chain definition files for each chain you create. As you create chains, their configuration files (in JSON format) will be placed in this folder.
- The **logs/** folder contains `app.log`, which algorithms can write to in order to provide information to a developer.
- `.env`: Environment variables needed by the project are stored in this file.
- `.gitignore`: This file will be familiar to you if you have used the git utility for managing source code. It tells git which files to leave out of a repository when saving or publishing it.
- `__init__.py`: This file tells Python that the folder should be considered a Python package.

- `config.py`: This is where you can set different parameters for the project, changing the way it works. See the [Configuring a Project](#) section below.
- `licenses.json`: This file is a placeholder for now, but will store license keys issued by developers for algorithms that require them.
- `run.py`: This file essentially links the project to the ATK.

1.4.1 Creating a project

The easiest way to create a project is to use the ATK CLI:

```
alg cp myproject
```

See the CLI docs for more information and examples.

1.4.2 Configuring a Project

`config.py`

The `config.py` file contains different project settings. A new project's `config.py` will look like this:

```
import logging
import os
import sys

from logging import Formatter
from logging.handlers import RotatingFileHandler

SECRET_KEY = os.environ['FLASK_SECRET_KEY']
ATK_PATH = os.path.dirname(os.path.abspath(__file__))
API_KEY = os.environ['ATK_API_KEY']

sys.path.append(ATK_PATH)
dirname = os.path.dirname
logfile = os.path.join(dirname(__file__), 'logs', 'app.log')
handler = RotatingFileHandler(logfile, maxBytes=10240, backupCount=10)
handler.setLevel(logging.DEBUG)
handler.setFormatter(Formatter(
    '%(asctime)s %(levelname)s %(message)s [in %(pathname)s:%(lineno)d]'
))
# Add additional logger handlers here, add to LOG_HANDLERS list

LOG_HANDLERS = [handler, ]

# Add your custom settings below
```

These settings should probably not be changed, but you can add your own as the comment at the bottom of the file indicates.

About logging

As you can see from the configuration here, the `LogLevel` by default is set to `DEBUG`. If you want to write more information to your log file (located in **logs/app.log** in your project), then keep this at `DEBUG`. If you want to write

less - such as when you're in a production setting - you can raise this to something like `WARNING` or `ERROR`. Just change it like so:

```
handler.setLevel(logging.WARNING)
```

See [Python's documentation](#) on logging for more details.

Optional settings

Here are some additional settings you can add to the project using `config.py`:

CORS_ORIGIN_WHITELIST

You probably will never need to adjust this setting, but if you create an application that uses the ATK (e.g.: to run a chain from another application on the web), then that application's URL needs to be in this list. This is a safety measure to prevent unwanted apps from hitting your site.

Default: `[]`

Example:

```
CORS_ORIGIN_WHITELIST = ['http://mysite.com', ]
```

DEFAULT_WORKING_ROOT

When a chain runs, an algorithm can use this folder to store temporary files as well as file-based results. Each chain gets a unique ID, and the ATK makes a folder under `DEFAULT_WORKING_ROOT` with that ID as its name. Within that folder, a folder named **temp** and one named **results** are also created. Anything stored in **temp** gets deleted after a chain finishes, but files in **results** remain and can be used later.

Default: `'/tmp'`

Example:

```
DEFAULT_WORKING_ROOT = '/users/myusername/atk'
```

.env

`.env` is a special configuration file. Anything placed in this file gets added to the system environment variables when the ATK is running. This is a place to store information that will be different from one environment to another (e.g.: one set of variables for development, one for production). Because of this, the `.env` file is listed in `.gitignore` and will not be added to a git repository if you create one. It's also a place to store information you don't want anyone to see (like security tokens).

A new project will have a `.env` file that looks like this:

```
FLASK_SECRET_KEY=
↪ "*****"
ATK_API_KEY="*****"
ATK_MANAGEMENT_API_KEY="*****"
FLASK_ENV=development
```

The “*”s will be random characters generated when you create the project using the CLI.

Notice that this is not a Python program, but just a text file. Here is an explanation of the parameters:

FLASK_SECRET_KEY

This is a key used internally by Flask to protect data submitted through forms and also to sign cookies. This key can contain unicode characters.

To be on the safe side, do not change this value

ATK_API_KEY

This key is used when running a chain. You will paste it into the Test Run form when testing your chains, and use it when you run a chain from an external program. It's also used when querying the ATK about what chains and algorithms are installed.

ATK_MANAGEMENT_API_KEY

This key is used to find out information about the ATK node, like how much load the system has or to retrieve the application log file. This key must be different than the ATK_API_KEY. If you don't want to enable these features, you can remove this line from the `.env` file.

Note: If you use TileDriver Process™, removing this key will reduce functionality

FLASK_ENV

This is another Flask internal configuration setting. The two recognized options are “development” and “production”. The development environment enables “DEBUG” in Flask automatically, which provides you the developer with useful information when testing out your code.

Also, the development environment itself cannot be accessed when this is set to “production”.

Note: You do not use quotation marks for this setting. The line would be:

```
FLASK_ENV=production
```

if you wanted to use the production environment.

1.5 The ATK Development Environment

When you work on algorithm projects, you can use the ATK's web-based development environment to make your life easier.

1.5.1 Running the environment

Once you have created a project, you can use:

```
alg run
```

to start the development environment. This command starts a basic web server and provides access to API endpoints within the ATK that allow you to create and edit algorithms and processing chains, and test your chains to see if they work.

Note: If you are running the ATK virtual server, such as with VirtualBox, you may need to run the development environment with the `--host` parameter as follows:

```
alg run --host=0.0.0.0
```

After you use `alg run`, you can point your web browser to <http://localhost:5000/> and you will see the screen below:

It's Working!

Congratulations! You created an Algorithm Toolkit (ATK) project. This web browser interface can help you navigate your project and make changes to it; however, you're free to modify source files or use the CLI directly if you prefer.

Where to go from here?

Now that you are up and running, here are a few things you might do next:

- [Create a new algorithm](#)
- [Create a new chain of algorithms, or make changes to an existing one](#)
- [Brush up on the documentation](#)

If you built this using the example project, or if you have created your own algorithm chains, you can use the "Test Run" menu at the top of this page to try them out.

About BeamIO

The ATK was designed and built by BeamIO, an innovative data processing company.

[Learn More »](#)

TileDriver

Looking for an easy yet powerful way to manage and visualize geospatial data and imagery? Try our TileDriver cloud-based suite. A Basic plan with 5 layers and 2.5GB of storage is free.

[Check it out »](#)

Algorithm Marketplace

Installing the ATK is just the beginning. The Marketplace has hundreds of algorithms you can use in your project.

[Go there »](#)

© You 2019



Under the hood

When you use `alg run` from a Terminal window, you will see lines like the following appear:

```
* Serving Flask app "run" (lazy loading)
* Environment: development
* Debug mode: on
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger PIN: 225-327-370
```

If you've worked with the web microframework [Flask](#), this will look familiar. Each algorithm project is a Flask app.

Here's what's happening:

- The name of the app is "run" (the `run.py` module in your project)
- You're in "development" mode
- "Debug" is enabled
- The address to which to point your browser is <http://127.0.0.1:5000/> (you can also use `localhost`, which is easier to remember)
- It also gives you a debugger PIN code, which will be handy later (see the Debugging section)

1.5.2 Stopping The Environment

From the terminal window, you can hit `^C` on your keyboard to stop the development server. On the web interface, the home page has a Stop button in the footer (see screen shot above); this can be useful when it seems as though your development server will not shut down.

1.5.3 The Project Homepage

The first page you come to (shown above) is the project homepage (naturally enough). This gives you some helpful hints and links about what you can do with the ATK. There are also other links to related sites you may find useful.

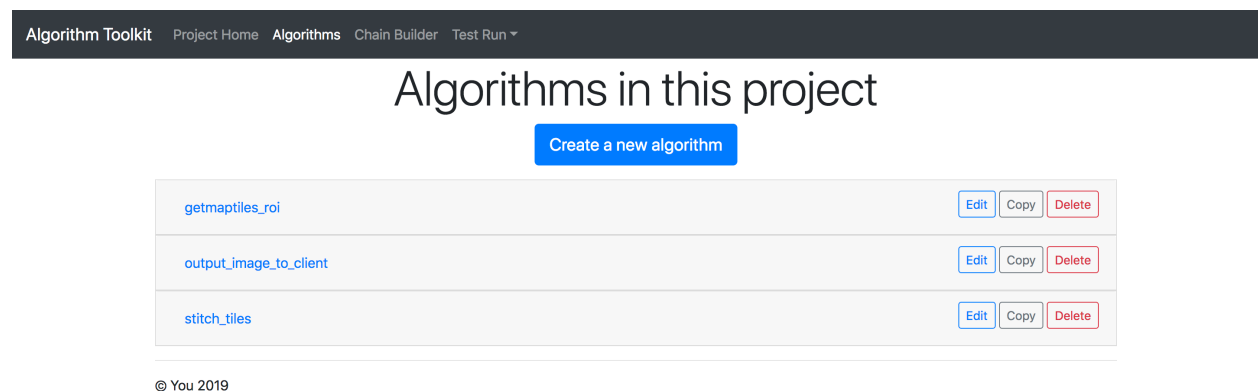
Across the top of the screen is the ATK menu bar. It provides access to these areas of the ATK development environment:

- **Project Home** (the current page)
- **Algorithms**: A listing of the algorithms installed in this project, and information about them
- **Chain Builder**: A tool to link algorithms together in processing chains
- **Documentation**: A link to this site, or to local docs if you created the project using the `–with-docs` flag
- **Test Run**: A test harness for running algorithm chains

Let's go over each of these pages in turn.

1.5.4 Algorithms Page

For a new project, this page will be empty except for a link to create an algorithm. If you installed the example project, you'll see this screen:



You can click on the name of an algorithm to get more information about it.

getmaptiles_roi Edit Copy Delete

Get Map Tiles In ROI

This algorithm will gather up map tiles at a given zoom level that intersect with the provided polygon. The source is the national map provided by USGS. All tiles will be written out to disk at a specified location. This location is also saved onto the chain ledger.

Version: 0.0.1
License: MIT
Homepage: <https://tiledriver.com/developer>

Parameters:

Name	Description	Required
roi	Polygon WKT to obtain tiles that intersect.	Yes
zoom	Zoom level	Yes
cache_path	Path to save gathered tiles to.	Yes

Outputs:

Name	Description
image_chips_dir	Local path to directory of map tiles

See the section *Working With Algorithms* for more details about what this page does.

1.5.5 Chain Builder Page

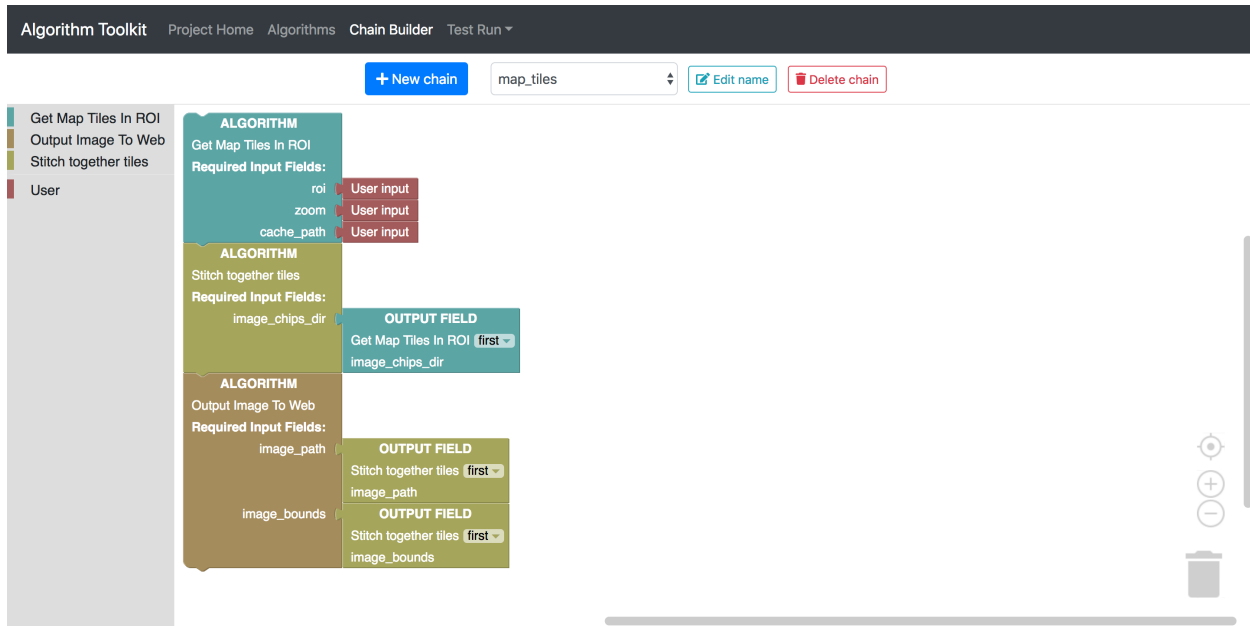
If you installed the example project, you'll see this page:

Algorithm Toolkit Project Home Algorithms Chain Builder Test Run

+ New chain
Select a chain to view/edit
Edit name
Delete chain

- Get Map Tiles In ROI
- Output Image To Web
- Stitch together tiles
- User

The example project comes with one chain, called “map_tiles”. You can select it from the drop-down menu that reads “Select a chain to view/edit”. When you select a chain, the chain definition is displayed:



See the section Building Chains for more details on this feature.

1.5.6 Test Run Page

Each chain you create becomes an option under the “Test Run” menu in the top navigation bar. In the example project, selecting the “map_tiles” chain displays this screen:

Enter parameters for each algorithm in the chain

API Key*

Fields marked with a * are required

Get Map Tiles In ROI

Polygon WKT*

Enter well known text (WKT) for the polygon region to obtain intersecting tiles.

Zoom level*

Enter an integer corresponding to the zoom level, 16 is max value supported.

Tile Cache Location*

Absolute path to directory for saving tiles

Stitch together tiles

No input needed.

Output Image To Web

No input needed.

Run Algorithm Chain

The form on this page allows you to run the chain by inputting various parameters and clicking “Run Algorithm Chain”. See the section Testing Algorithm Chains for more details on how to use the test harness.

1.6 Working With Algorithms

1.6.1 What is an algorithm?

In terms of the ATK, an algorithm is a Python package that contains at least some Python code that follows a specific structure, and a JSON file describing the algorithm's input and outputs. Like any Python package, an algorithm can contain a lot of other things as well.

1.6.2 Creating an Algorithm

There are three ways you can create an algorithm:

1. Use the development environment web interface (see below section)
2. Use the CLI (see the CLI section in the docs for the `ca` command)
3. Manually create it yourself

We recommend the first two options over the third. We've created a lot of checks to make sure things like typos and missing files don't happen.

1.6.3 Structure Of An Algorithm

When you create an algorithm using the CLI or the development environment web interface, the following five files will be created inside the `algorithms/algorithm_name` folder in your project:

```
__init__.py
algorithm.json
LICENSE
main.py
README.md
test.py
```

Let's look at each of these in turn.

- `__init__.py`: This file tells Python to consider this folder a Python package; it will probably never have anything in it
- `algorithm.json`: This file contains the algorithm's configuration, including any input parameters or data that the algorithm outputs
- `LICENSE`: This file contains whatever licensing terms you want to place on the algorithm
- `main.py`: This is where you will write the code that your algorithm runs, or call code you write in other Python programs
- `README.md`: A helpful document for users of your algorithm
- `test.py`: A module for your algorithm's unit tests

Most of the time you will edit only `main.py` (and probably `test.py`) directly. The development environment web interface handles updates to the other files for you.

`algorithm.json`

Even though the web interface updates this file on your behalf as you change your algorithm, you can edit it yourself if you wish. Here's what one looks like:

```

{
    "name": "getmaptiles_roi",
    "display_name": "Get Map Tiles In ROI",
    "description": "This algorithm will gather up map tiles at a given zoom level_
↳that inteseect with the provided polygon. The source is the national map provided by_
↳USGS. All tiles will be written out to disk at a specified location. This location_
↳is also saved onto the chain ledger.",
    "version": "0.0.1",
    "license": "MIT",
    "private": false,
    "homepage": "https://tiledriver.com/developer",
    "required_parameters": [
        {
            "name": "roi",
            "description": "Polygon WKT to obtain tiles that intersect.",
            "display_name": "Polygon WKT",
            "data_type": "string",
            "field_type": "text",
            "help_text": "Enter well known text (WKT) for the polygon_
↳region to obtain intersecting tiles.",
            "min_value": null,
            "max_value": null,
            "default_value": "POLYGON((-77.0419692993164 38.9933585922412,
↳-77.17311859130861 38.891887936025896,-77.03853607177736 38.790272111428706,-76.
↳91013336181642 38.891887936025896,-77.0419692993164 38.9933585922412))",
            "custom_validation": null,
            "parameter_choices": [],
            "sort_order": 0
        },
        {
            "name": "zoom",
            "description": "Zoom level",
            "display_name": "Zoom level",
            "data_type": "integer",
            "field_type": "number",
            "help_text": "Enter an integer corresponding to the zoom_
↳level, 16 is max value supported.",
            "min_value": 5,
            "max_value": 16,
            "default_value": 14,
            "custom_validation": null,
            "parameter_choices": [],
            "sort_order": 1
        },
        {
            "name": "cache_path",
            "description": "Path to save gathered tiles to.",
            "display_name": "Tile Cache Location",
            "data_type": "string",
            "field_type": "text",
            "help_text": "Absolute path to directory for saving tiles",
            "min_value": null,
            "max_value": null,
            "default_value": "/tmp/tiles/map_tiles",
            "custom_validation": null,
            "parameter_choices": [],
            "sort_order": 2
        }
    ]
}

```

(continues on next page)

(continued from previous page)

```

    },
    "optional_parameters": [],
    "outputs": [
        {
            "name": "image_chips_dir",
            "description": "Local path to directory of map tiles",
            "display_name": "Path To Tiles",
            "data_type": "string"
        }
    ]
}

```

Here's what each of the fields means:

Parameter	Description
name	A short name with no spaces, which must be unique across algorithms in your project
display_name	A more descriptive name that will be displayed to the user
description	A long description of what the algorithm does
version	The algorithm version number in text form
license	A description of the license terms, using the SPDX identifier (https://spdx.org/licenses/)
private	If the algorithm is published to the Algorithm Registry, setting private to true will cause it not to display in a Registry listing
homepage	A web address where more information can be found about the algorithm (could also be a link to a source code repository)

The algorithm.json file also defines input and output parameters. For inputs, there are “required_parameters” and “optional_parameters”. The list of outputs is just called “outputs”. Each parameter and each output has the same structure:

Input Parameters

Parameter	Description
name	A short name with no spaces, which must be unique across parameters in this algorithm
display_name	A more descriptive name that will be displayed to the user
description	A long description of what the parameter is for
data_type	The type of data the input parameter will accept
field_type	When displayed on a web form, which HTML field type to use
help_text	Text that would be displayed below a form field, providing guidance on how to enter a value
min_value	If a numeric field, what is the minimum value the input should accept
max_value	If a numeric field, what is the maximum value the input should accept
default_value	The value to use if not provided (also will display in a web form)
custom_validation	Custom validation rules (see below)
parameter_choices	A comma-separated list of values that the field will accept (often goes with a Select field type, but does not have to)
sort_order	On a web form, where should this field be displayed relative to other input fields

Outputs

Parameter	Description
name	A short name with no spaces, which must be unique across outputs of this algorithm
display_name	A more descriptive name that will be displayed to the user
description	A long description of what the output is
data_type	The type of data the output will produce

It may seem like a lot of information for you to fill out, but the more detail you enter into `algorithm.json` the easier it will be for another developer to use what you create. An important part of the ATK is code sharing and reuse among the developer community.

main.py

As stated earlier, most of your code will live in this file, or this file will point to other code you have written. When you create a new algorithm, the `main.py` will look like this:

```
from algorithm_toolkit import Algorithm, AlgorithmChain

class Main(Algorithm):

    def run(self):
        cl = self.cl # type: AlgorithmChain.ChainLedger
        params = self.params # type: dict
        # Add your algorithm code here

        # Do not edit below this line
        return cl
```

It may not look like much, but there's a lot going on in these few lines of code.

Every time your algorithm runs, this `run()` routine will be called. As you'll notice, this `Main` class inherits from a class inside the ATK called `Algorithm`. To see what the `Algorithm` class can do, check out the API section of these docs.

When this code runs, a dictionary called `params` will contain the data entered by a user or application calling the algorithm. In the `get_map_tiles` example above, `params` would look like this:

```
{
    'roi': 'POLYGON((-77.0419692993164 38.9933585922412,-77.17311859130861 38.
↪891887936025896,-77.03853607177736 38.790272111428706,-76.91013336181642 38.
↪891887936025896,-77.0419692993164 38.9933585922412))',
    'cache_path': '/tmp/tiles/map_tiles',
    'zoom': 14
}
```

If you wanted to get the value of the `zoom` parameter in your code, you could refer to it like this:

```
zoom = params['zoom']
```

This is how your algorithm can receive the data it needs to operate. Notice that the `zoom` parameter is an integer in the `params` dictionary? The ATK does some nice things for you, such as making sure the data will be in the proper format you need it to be in (via the `data_type` field in the input parameter). If it's not, the user will be alerted to the problem and your code won't even run. This means you can focus on making use of the data inputs rather than spending time validating them.

You also probably noticed a reference to a mysterious variable called simply `cl`. This stands for Chain Ledger, and is one of the most important aspects of the ATK. It's so important, *it has its own section in the docs*.

The short version is: the Chain Ledger is how you will provide outputs to other algorithms in the chain, and how you can see a history of everything the chain did after it runs. This is a hugely powerful and useful feature, because it gives you the ability to reproduce results at a later time.

You can put literally anything Python can create on the Chain Ledger. Most of the time, you will just place your algorithm's outputs on it. In the `get_map_tiles` example, the algorithm might do it like so:

```
cl.add_to_metadata('image_chips_dir', '/tmp/chips')
```

See the Chain Ledger section for more details on how this works.

1.6.4 Algorithm Input Validation

Data types

The `data_type` field can be set to any one of the following options:

- String (the default)
- Integer
- Float
- Array (entered as a comma-separated list in a form field; the list members can be any data type)

Before an algorithm is called, the ATK uses the `data_type` to ensure that the correct type is being sent to the algorithm. If the value can be cast to the correct type, the ATK will do that. Otherwise an error will be raised.

Validation controls

We said earlier that the ATK validates inputs before your code runs. In addition to the data type validation, you can set boundaries on what values will be accepted. Several fields within the parameter definition help with this: `min_value`, `max_value`, `parameter_choices`, and `custom_validation`. The first three are self-explanatory; let's look at `custom_validation`.

custom_validation

The ATK comes with a set of validators you can refer to in this field. They are:

greaterthan

The value must be greater than the value of another input parameter. Numeric inputs only.

Example:

```
"custom_validation": "greaterthan.another_input"
```

where "another_input" is the name of the parameter whose value that must be smaller than this value.

Note: `greaterthan` and `lessthan` should come in pairs. In other words, if `input1` should be greater than `input2`, `input1` should use `greaterthan.input2` as its custom validation, and `input2` should have `lessthan.input1` as its custom validation.

lessthan

The value must be less than the value of another input parameter. Numeric inputs only.

Example:

```
"custom_validation": "lessthan.another_input"
```

where “another_input” is the name of the parameter whose value that must be greater than this value.

Note: greaterthan and lessthan should come in pairs. In other words, if input1 should be less than input2, input1 should use `lessthan.input2` as its custom validation, and input2 should have `greaterthan.input1` as its custom validation.

evenonly

The value must be an even integer. The `data_type` must also be Integer.

Example:

```
"custom_validation": "evenonly"
```

oddonly

The value must be an odd integer. The `data_type` must also be Integer.

Example:

```
"custom_validation": "oddonly"
```

^

The value must conform to a regular expression pattern, defined after the caret (^) symbol. String inputs only.

Example:

```
"custom_validation": "^\d{3}-\d{3}-\d{4}$"
```

This would require the input to have a US phone number-like format. See the [Python documentation](#) for regular expression syntax.

Outputs

If your algorithm outputs a value, that value must also have a data type. When building chains, it’s important to think about what type of output your algorithm produces because otherwise the next algorithm in the chain might not get a value of the right data type (and thus would never run, as we stated earlier). In fact, the Chain Builder prevents you from linking an output of one algorithm to the input of another if they’re not the same data type (more on this in the Chain Builder section).

1.6.5 Using The Web Interface

When you click the Create a New Algorithm button on the Algorithms page, you will see the create/edit algorithm form:

Create/Edit an Algorithm

Algorithm Name
Enter a short name for the algorithm (no spaces)

Display Name
Enter a more descriptive name for the algorithm

Description

Version
Enter a version string (e.g.: "0.0.1", "beta")

Algorithm Website
Include a URL where someone could go for documentation

Private ☐
Make this algorithm unlisted (if publishing to the Algorithm Registry)

License
See <https://choosealicense.com/licenses/> for info on various open source licenses

Algorithm Input Parameters:

Name	Description	Required
------	-------------	----------

Algorithm Outputs:

Name	Description
------	-------------

© You 2019

If you pull up the `get_map_tiles` algorithm, you can see the form filled out with the data from its `algorithm.json`:

Create/Edit an Algorithm

Algorithm Name
Enter a short name for the algorithm (no spaces)

Display Name
Enter a more descriptive name for the algorithm

Description

This algorithm will gather up map tiles at a given zoom level that intersect with the provided polygon. The source is the national map provided by USGS. All tiles will be written out to disk at a specified location. This location is also saved onto the chain ledger.

Version
Enter a version string (e.g.: "0.0.1", "beta")

Algorithm Website
Include a URL where someone could go for documentation

Private ☐
Make this algorithm unlisted (if publishing to the Algorithm Registry)

License
See <https://choosealicense.com/licenses/> for info on various open source licenses

Algorithm Input Parameters: Add a parameter

Name	Description	Required
roi	Polygon WKT to obtain tiles that intersect.	Yes
zoom	Zoom level	Yes
cache_path	Path to save gathered tiles to.	Yes

Algorithm Outputs: Add an output

Name	Description
image_chips_dir	Local path to directory of map tiles

Save Algorithm Cancel

☐ Also update README file

© You 2019

Only the algorithm name and display name fields are required, but the more information you provide the easier it will be for another developer to use your algorithm.







License

We provide several open source licenses that can be applied to your algorithm. When you click the license drop-down menu, you'll see the following options:

Private ☐ Make this algorithm unlisted (if publishing to the Algorithm Registry)

License

☒ MIT
 BSD-3-Clause
 GNU AGPLv3
 GNU GPLv3
 GNU LGPLv3
 Mozilla
 Apache
 The Unlicense
 Proprietary
 See LICENSE File

Name			
roi			
zoom	Zoom level	Yes	  
cache_path	Path to save gathered tiles to.	Yes	  

Algorithm Outputs: Add an output

When you choose a license, a LICENSE file will be placed in your algorithm's folder corresponding to the open source license version you select. The text of these licenses come from the Open Source Initiative website (<https://opensource.org/licenses>).

If you do not wish to apply an open source license, you can choose one of two additional options: "Proprietary" or "See LICENSE File". Both of these options will create a blank LICENSE file in your algorithm's folder, and you can add whatever terms you deem appropriate.

Adding and editing parameters

If you click the Edit button (looks like a pencil) for the `roi` parameter, you will see the create/edit input parameter form slide in from the right:

Algorithm Toolkit

Project Home

Algorithms

Chain Builder

Test Run

Create/Edit an Algorithm

Algorithm Name

getmaptiles_roi

Enter a short name for the algorithm (no spaces)

Display Name

Get Map Tiles In ROI

Enter a more descriptive name for the algorithm

Description

This algorithm will gather up map tiles at a given zoom level that intersect with the provided by USGS. All tiles will be written out to disk at a specified location. This

Version

0.0.1

Enter a version string (e.g.: "0.0.1", "beta")

Algorithm Website

https://tiledriver.com/developer

Include a URL where someone could go for documentation

Private

☐

Make this algorithm unlisted (if publishing to the Algorithm Registry)

License

MIT

See <https://choosealicense.com/licenses/> for info on various open source licenses

Algorithm Input Parameters:

Name	Description
roi	Polygon WKT to obtain tiles that intersect.
zoom	Zoom level
cache_path	Path to save gathered tiles to.

Algorithm Outputs:

Name	Description
image_chips_dir	Local path to directory of map tiles

Save Algorithm

Cancel

☐ Also update README file

© You 2019

Edit Input Parameter

Parameter Name

roi

Enter a unique short name for the parameter (no spaces)

Required

☒

Require a value for this parameter

Display Name

Polygon WKT

Enter a more descriptive name for the parameter

Description

Polygon WKT to obtain tiles that intersect.

Data Type

String

Select the type of data this parameter will accept

Field Type

Text

Select the type of form field that would be displayed for this parameter

Help Text

Add a parameter

Enter well known text (WKT) for the polygon region to obtain intersecting tiles.

Enter some information that would help a user know how to enter a value for this parameter

Minimum Value

Set the smallest value of this parameter (number parameter types only)

Maximum Value

Set the largest value of this parameter (number parameter types only)

Default Value

POLYGON((-77.0419692993164 38.9933585922412,-77.1731185913C

Set the default value of this parameter

Custom Validation

Numeric parameters (integers and floats) look like this:

Algorithm Toolkit
Project Home
Algorithms
Chain Builder
Test Run

Create/Edit an Algorithm

Algorithm Name

Enter a short name for the algorithm (no spaces)

Display Name

Enter a more descriptive name for the algorithm

Description

Version

Enter a version string (e.g.: "0.0.1", "beta")

Algorithm Website

Include a URL where someone could go for documentation

Private

☐

Make this algorithm unlisted (if publishing to the Algorithm Registry)

License

See <https://choosealicense.com/licenses/> for info on various open source licenses

Algorithm Input Parameters:

Name	Description
roi	Polygon WKT to obtain tiles that intersect.
zoom	Zoom level
cache_path	Path to save gathered tiles to.

Algorithm Outputs:

Name	Description
image_chips_dir	Local path to directory of map tiles

☐ Also update README file

Edit Input Parameter

Parameter Name

Enter a unique short name for the parameter (no spaces)

Required

☒

Require a value for this parameter

Display Name

Enter a more descriptive name for the parameter

Description

Data Type

Select the type of data this parameter will accept

Field Type

Select the type of form field that would be displayed for this parameter

Help Text

Enter some information that would help a user know how to enter a value for this parameter

Minimum Value

Set the smallest value of this parameter (number parameter types only)

Maximum Value

Set the largest value of this parameter (number parameter types only)

Default Value

Set the default value of this parameter

Custom Validation

© You 2019

Adding and editing Outputs

If you click the Edit button (looks like a pencil) for the `image_chips_dir` output, you will see the create/edit output form slide in from the right:

Algorithm Toolkit
Project Home
Algorithms
Chain Builder
Test Run

Create/Edit an Algorithm

Algorithm Name

Enter a short name for the algorithm (no spaces)

Display Name

Enter a more descriptive name for the algorithm

Description

This algorithm will gather up map tiles at a given zoom level that intersect with the provided by USGS. All tiles will be written out to disk at a specified location. This source is the national map.

Version

Enter a version string (e.g.: "0.0.1", "beta")

Algorithm Website

Include a URL where someone could go for documentation

Private

☐

Make this algorithm unlisted (if publishing to the Algorithm Registry)

License

See <https://choosealicense.com/licenses/> for info on various open source licenses

Algorithm Input Parameters:

Name	Description	Required
roi	Polygon WKT to obtain tiles that intersect.	Yes
zoom	Zoom level	Yes
cache_path	Path to save gathered tiles to.	Yes

Algorithm Outputs:

Name	Description
image_chips_dir	Local path to directory of map tiles

☐ Also update README file

Edit Output

Output Name

Enter a unique short name for the output (no spaces)

Display Name

Enter a more descriptive name for the output

Description

Local path to directory of map tiles

Data Type

Select the type of data this output will produce

© You 2019

Head over to *Creating Your First Algorithm* to see this form in action.

1.7 Creating Your First Algorithm

Let's dive into algorithm development with everyone's favorite trope: the Hello World program!

We'll use the development environment web interface to set up the algorithm, then modify the code in `main.py` to enhance the functionality of our algorithm.

1.7.1 Creating a Project

We'll need an algorithm project to start things off. Take a look at the *Installation* documentation for how to create a new project with the example project included. Having the example available is a great reference for how to make your own algorithms.

1.7.2 Setting Things Up

As you'll recall from the last page, the create new algorithm form looks like this (click the Create a New Algorithm button from the Algorithms page):

Algorithm Toolkit
Project Home
Algorithms
Chain Builder
Test Run

Create/Edit an Algorithm

Algorithm Name

Enter a short name for the algorithm (no spaces)

Display Name

Enter a more descriptive name for the algorithm

Description

Version

Enter a version string (e.g.: "0.0.1", "beta")

Algorithm Website

Include a URL where someone could go for documentation

Private

☐

Make this algorithm unlisted (if publishing to the Algorithm Registry)

License

See <https://choosealicense.com/licenses/> for info on various open source licenses

Algorithm Input Parameters:

Name

Description

Required

Algorithm Outputs:

Name

Description

Save Algorithm

Cancel

© You 2019

Let's fill in some details about our algorithm. We'll name it "my_first_algorithm", give it a brief description, and choose the MIT license (we're feeling generous today):

Algorithm Toolkit

Project Home

Algorithms

Chain Builder

Test Run

Create/Edit an Algorithm

Algorithm Name

my_first_algorithm

Enter a short name for the algorithm (no spaces)

Display Name

My First Algorithm

Enter a more descriptive name for the algorithm

Description

This is my first algorithm, and I'm proud of it.

Version

0.0.1

Enter a version string (e.g.: "0.0.1", "beta")

Algorithm Website

Include a URL where someone could go for documentation

Private

☐

Make this algorithm unlisted (if publishing to the Algorithm Registry)

License

MIT

See <https://choosealicense.com/licenses/> for info on various open source licenses

Algorithm Input Parameters:

Name	Description	Required
------	-------------	----------

Add a parameter

Algorithm Outputs:

Name	Description
------	-------------

Add an output

Save Algorithm

Cancel

© You 2019

Click “Save Algorithm” and you’ll see your algorithm appear in the list:

Algorithms in this project

Create a new algorithm

getmaptiles_roi	Edit Copy Delete										
my_first_algorithm	Edit Copy Delete										
<p>My First Algorithm</p> <p>This is my first algorithm, and I'm proud of it.</p> <p>Version: 0.0.1 License: MIT Homepage:</p> <p>Parameters:</p> <table border="1"> <thead> <tr> <th>Name</th> <th>Description</th> <th>Required</th> </tr> </thead> <tbody> <tr> <td colspan="3"> </td> </tr> </tbody> </table> <p>Outputs:</p> <table border="1"> <thead> <tr> <th>Name</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td colspan="2"> </td> </tr> </tbody> </table>		Name	Description	Required				Name	Description		
Name	Description	Required									
Name	Description										
output_image_to_client	Edit Copy Delete										
stitch_tiles	Edit Copy Delete										

© You 2019

That's it! Notice we did not even have to name any inputs or outputs in order to create the algorithm. It will be more useful once we have those, but for now let's leave it at that.

1.7.3 What Just Happened?

After we clicked the Save button, the ATK created a new folder in our project called **algorithms/my_first_algorithm**. Within that folder, you'll find the same files described in *Working With Algorithms*:

```
__init__.py
algorithm.json
LICENSE
main.py
README.md
test.py
```

The algorithm.json file contains the basic information you entered:

```
{
  "description": "This is my first algorithm, and I'm proud of it.",
  "display_name": "My First Algorithm",
  "homepage": "",
  "license": "MIT",
  "name": "my_first_algorithm",
  "optional_parameters": [],
  "outputs": [],
  "private": false,
  "required_parameters": [],
  "version": "0.0.1"
}
```

1.7.4 Now What?

We have the basic structure of an algorithm; now we need to make it do something.

Open up the `main.py` file in your new algorithm folder, using your favorite text editor or programming tool. You'll see the following code:

```
from algorithm_toolkit import Algorithm, AlgorithmChain

class Main(Algorithm):

    def run(self):
        cl = self.cl # type: AlgorithmChain.ChainLedger
        params = self.params # type: dict
        # Add your algorithm code here

        # Do not edit below this line
        return cl
```

Working With Algorithms describes the basics of what's going on here.

In order to make our Hello World program complete, we need to have it greet the world with its friendly message. To send a message to the user or calling function, we need to make use of the Chain Ledger. You'll recall from *Working With Algorithms* that the Chain Ledger is a Python dictionary, and you can access it using the variable `cl` in your algorithm.

We'll use a special key the Chain Ledger maintains: `chain_output_value`. Essentially, this key outputs a message or other result to the client. The output value can be unstructured text, JSON, CSV, or even binary files. For our program we'll just use text. Here's how you would output the famous greeting in your algorithm:

```
# Add your algorithm code here

chain_output = {
    'output_type': 'text',
    'output_value': 'Hello World!'
}
cl.add_to_metadata('chain_output_value', chain_output)

# Do not edit below this line
return cl
```

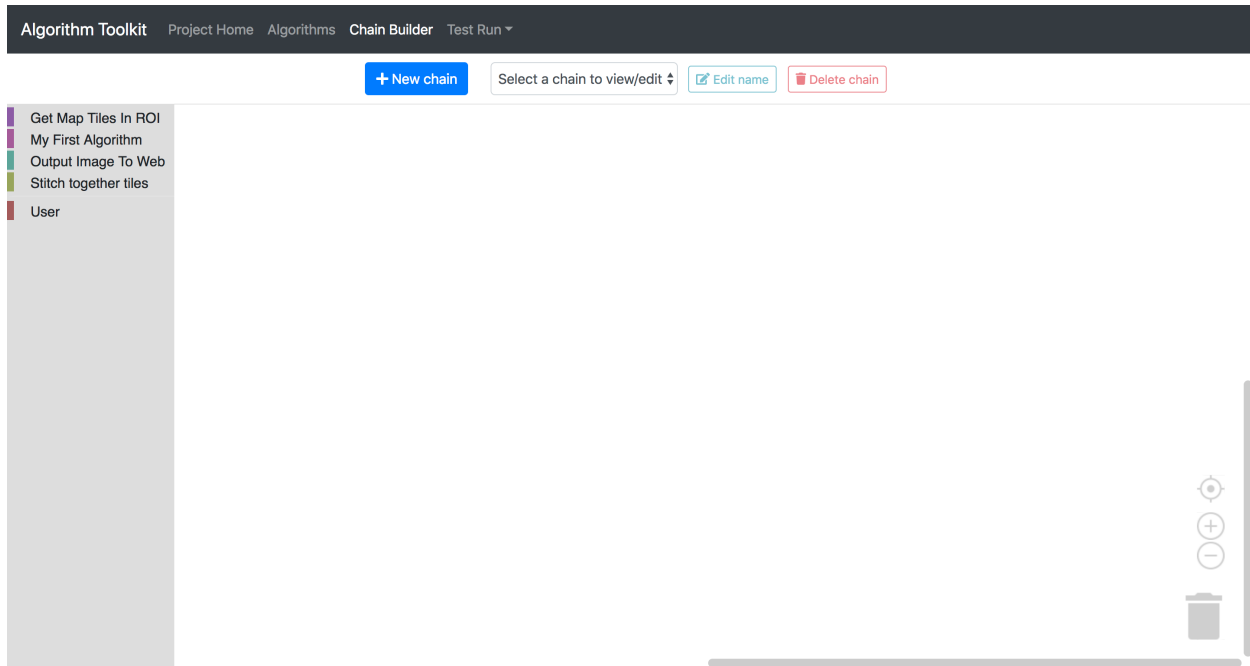
As you can see, `chain_output_value` is itself a dictionary with two keys: `output_type` ("text" in this case) and `output_value` ("Hello World!"). You use the Chain Ledger's `add_to_metadata` function to set the value of this key, and the function takes two arguments: the key name ("chain_output_value") and the key value (our dictionary).

Save the `main.py` file.

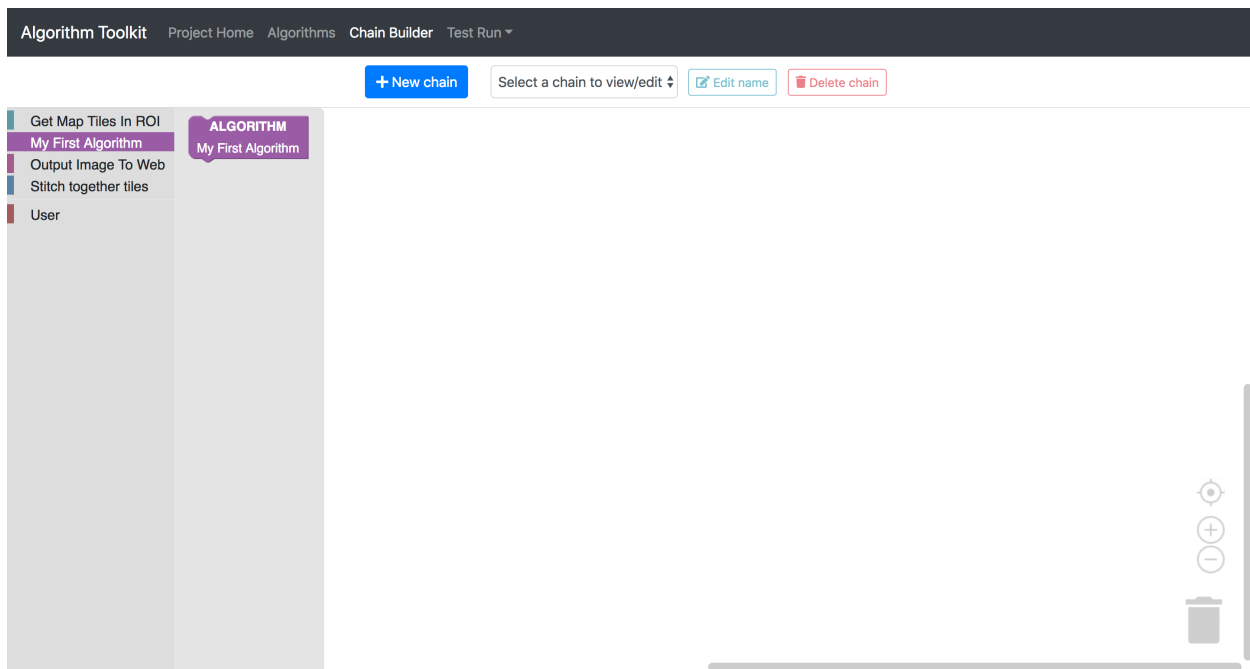
1.7.5 Running The Algorithm

In order to run this code, we need to set up a basic chain. The ATK does not run individual algorithms: it requires a chain, and once you create one you can use the Test Run harness provided in the development environment.

In the web interface, click the link for Chain Builder. You'll see your algorithm listed along with the example project algorithms:



When you click the name of your algorithm, you'll see a block representing it:



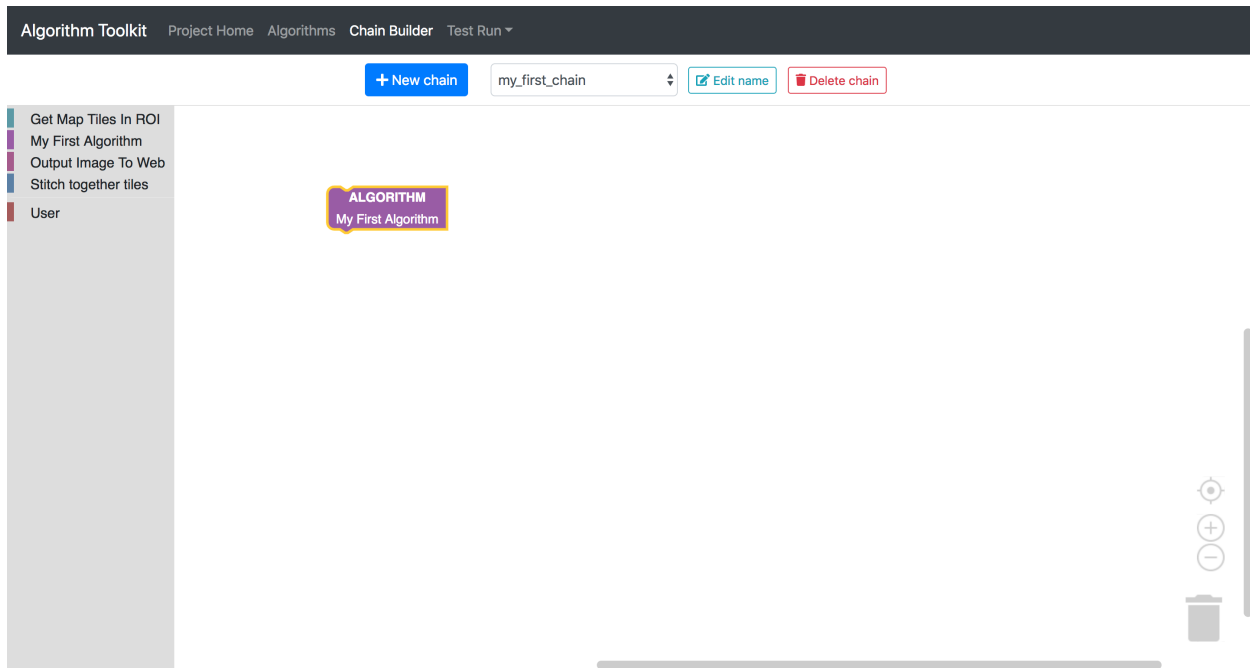
Drag the block to the right onto the Chain Canvas. You'll get a prompt to give this chain a name:

localhost:5000 says

Give your chain a name:

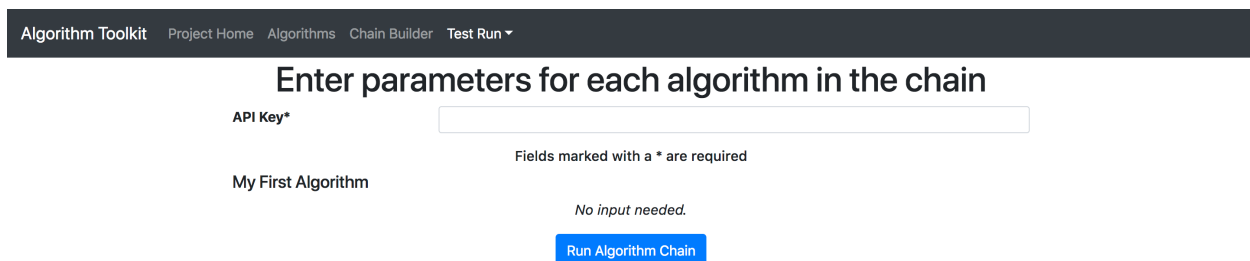
Cancel OK

After that you'll see your chain, with your single algorithm:



Believe it or not, that's it! However modest, you now have an algorithm processing chain.

The chain name will now appear in the Test Run menu at the top of the screen. Select it, and you'll see a basic web form:



Because we don't have any input parameters, there's not much here. You will however need your API key.

What is my API key?

The API key is a simple security mechanism to make sure no one runs your chain or does anything with your algorithm project unless you want them to. To find your key, open the `.env` file in your algorithm project, and copy the value in `ATK_API_KEY` (don't copy the quotation marks).

Paste the value into the API Key form field. Then click “Run Algorithm Chain”.

This chain will run pretty fast, since you're just outputting text to the screen. After it's finished, you'll see this:

The screenshot shows the 'Enter parameters for each algorithm in the chain' page. At the top, there's a navigation bar with 'Algorithm Toolkit', 'Project Home', 'Algorithms', 'Chain Builder', and 'Test Run'. The main heading is 'Enter parameters for each algorithm in the chain'. Below this, there's a form for 'API Key*' with a text input field. A note says 'Fields marked with a * are required'. Underneath, it says 'My First Algorithm' and 'No input needed.' There's a blue button labeled 'Run Algorithm Chain'. Below the button, 'Chain Status:' is shown with two progress bars: 'Algorithm Progress:' and 'Chain Progress:', both at 100%. A text box below the progress bars says 'Chain run complete'. At the bottom, 'Chain Result:' is shown with the text 'Hello World!'.

There it is! Our welcome message to the world.

1.7.6 Adding An Input

Now that we have a working algorithm and chain, we begin to see all kinds of possibilities. Let's start by allowing our greeting to be more personal.

Edit your algorithm in the web interface by clicking the Edit button next to its name in the Algorithms page. Then click Add Parameter.

We'll add a required input for the name of the person we want to greet. Fill out the relevant fields like this:

Algorithm Toolkit
Project Home
Algorithms
Chain Builder
Test Run

Create/Edit an Algorithm

Algorithm Name
my_first_algorithm
Enter a short name for the algorithm (no spaces)

Display Name
My First Algorithm
Enter a more descriptive name for the algorithm

Description
This is my first algorithm, and I'm proud of it.

Version
0.0.1
Enter a version string (e.g.: "0.0.1", "beta")

Algorithm Website
Include a URL where someone could go for documentation

Private
☐
Make this algorithm unlisted (if publishing to the Algorithm Registry)

License
MIT
See <https://choosealicense.com/licenses/> for info on various open source licenses

Algorithm Input Parameters:

Name	Description
------	-------------

Algorithm Outputs:

Name	Description
------	-------------

Save Algorithm
Cancel

☐ Also update README file

Create Input Parameter

Parameter Name
greeting_name
Enter a unique short name for the parameter (no spaces)

Required
☒
Require a value for this parameter

Display Name
Name of person to greet
Enter a more descriptive name for the parameter

Description
What should I call you?

Data Type
String
Select the type of data this parameter will accept

Field Type
Text
Select the type of form field that would be displayed for this parameter

Help Text
Please enter your full name
Enter some information that would help a user know how to enter a value for this parameter

Minimum Value
Set the smallest value of this parameter (number parameter types only)

Maximum Value
Set the largest value of this parameter (number parameter types only)


Click the Save button on the parameter form. You'll see the parameter listed:

36

Chapter 1. Contents

Create/Edit an Algorithm

Algorithm Name
Enter a short name for the algorithm (no spaces)

Display Name 
Enter a more descriptive name for the algorithm

Description

This is my first algorithm, and I'm proud of it.




Version
Enter a version string (e.g.: "0.0.1", "beta")

Algorithm Website
Include a URL where someone could go for documentation

Private ☐
Make this algorithm unlisted (if publishing to the Algorithm Registry)

License
See <https://choosealicense.com/licenses/> for info on various open source licenses

Algorithm Input Parameters: Add a parameter

Name	Description	Required	
 greeting_name	What should I call you?	Yes	 

Algorithm Outputs: Add an output

Name	Description
------	-------------

Save Algorithm Cancel

☐ Also update README file

© You 2019

Now click Save Algorithm.

If you open your algorithm.json file, you'll see the new parameter has been added to the definition:

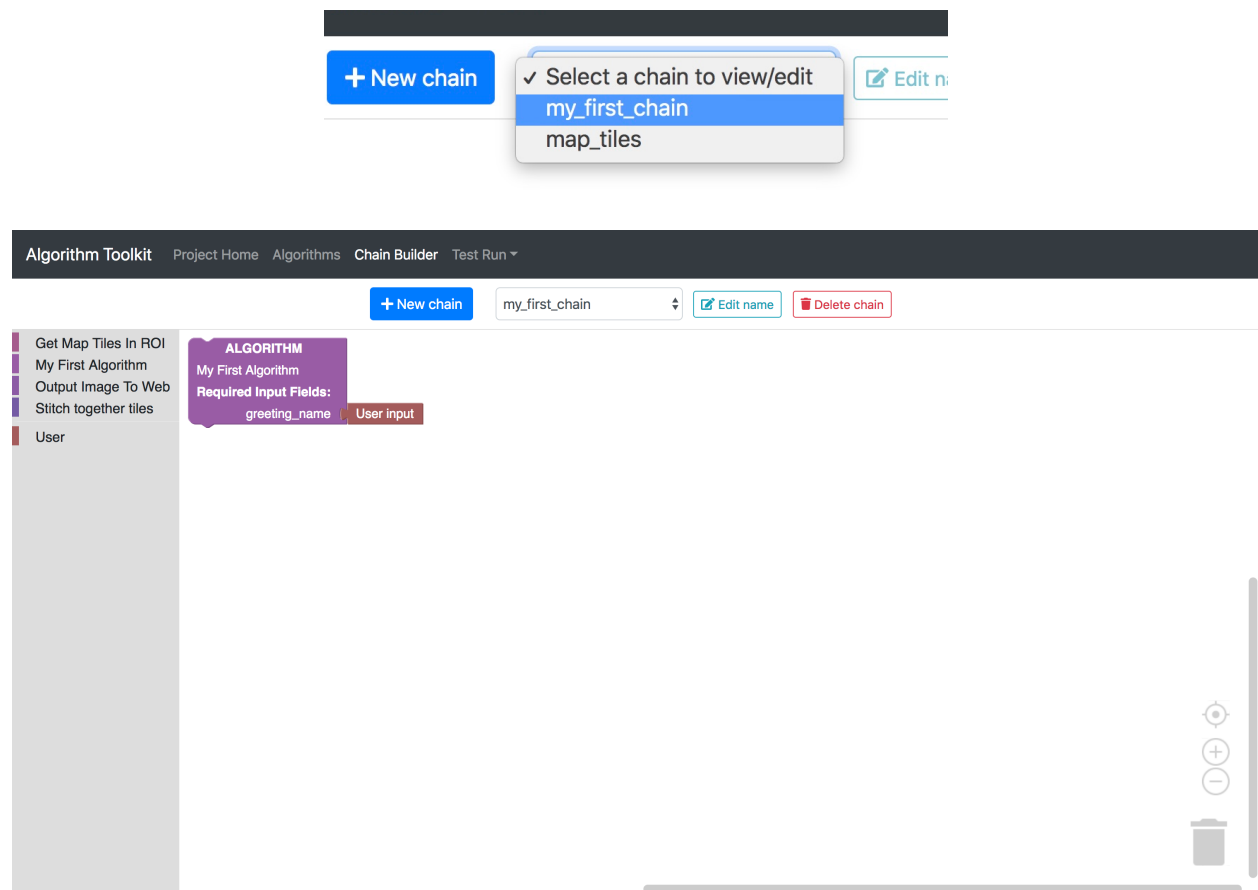
```
{
  "description": "This is my first algorithm, and I'm proud of it.",
  "display_name": "My First Algorithm",
  "homepage": "",
  "license": "MIT",
  "name": "my_first_algorithm",
  "optional_parameters": [],
  "outputs": [],
  "private": false,
  "required_parameters": [
    {
      "custom_validation": "",
      "data_type": "string",
      "default_value": "",
      "description": "What should I call you?",
      "display_name": "Name of person to greet",
      "field_type": "text",
      "help_text": "Please enter your full name",
      "max_value": "",
      "min_value": "",
```

(continues on next page)

(continued from previous page)

```
    "name": "greeting_name",
    "parameter_choices": "",
    "required": true,
    "sort_order": 0
  },
  {
    "version": "0.0.1"
  }
}
```

Since we added an input parameter, we need to modify the chain to tell the ATK how the input will get a value. Click Chain Builder, then select your chain from the drop-down menu:



Look at that! The ATK figured out that you wanted the user to input a value for the new field without you having to do anything. Awesome!

There are other options for how an input gets its value, which we'll explore in a later section. For now, let's run this chain using Test Run again:

Algorithm Toolkit
Project Home
Algorithms
Chain Builder
Test Run

Enter parameters for each algorithm in the chain

API Key*

Fields marked with a * are required

My First Algorithm

Name of person to greet* ⓘ

Please enter your full name

Run Algorithm Chain

You'll see the new input parameter on the form, along with the help text explaining what the field is for. If you hover over the info icon to the right, you'll see the description you entered for the parameter:

Each algorithm in the chain

Fields marked with a * are required

What should I call you? ⓘ

Run Algorithm Chain

Click Run Algorithm Chain.

Algorithm Toolkit Project Home Algorithms Chain Builder Test Run ▾

Enter parameters for each algorithm in the chain

API Key*

Fields marked with a * are required

My First Algorithm

Name of person to greet* ⓘ

Please enter your full name

[Run Algorithm Chain](#)

Chain Status:

Algorithm Progress:

100%

Chain Progress:

100%

Chain run complete

Chain Result:

Hello World!

Huh, you got the same result. How come?

1.7.7 Using The Input

Your algorithm code needs to do something with the input value. Let's open `main.py` again and change the code like this:

```
# Add your algorithm code here

name = params['greeting_name']
msg = 'Hello, %s!' % name

chain_output = {
    'output_type': 'text',
    'output_value': msg
}
cl.add_to_metadata('chain_output_value', chain_output)

# Do not edit below this line
return cl
```

Here you see we're using the `params` dictionary to take the parameter called "greeting_name" and assign it to a variable. The ATK saves the user input in that key in `params`. While you're developing, it can be helpful to open the `algorithm.json` file if you forget what you named a parameter.

Then we create the new greeting and assign it to the "output_value" for placing on the Chain Ledger. Looks like we're done here.

Save the changes to `main.py`

Note: The development environment can detect some changes to your algorithm code and restart itself so you don't have to remember to do it. When that happens, you'll see lines like this in your Terminal window:

```
* Detected change in '/myproject/algorithms/my_first_algorithm/main.py', reloading
* Restarting with stat
* Debugger is active!
* Debugger PIN: 225-327-370
```

Now go back to Test Run and run your chain. This time you'll see the right greeting:

The screenshot shows the 'Test Run' tab of the Algorithm Toolkit. At the top, a dark navigation bar contains links: 'Algorithm Toolkit', 'Project Home', 'Algorithms', 'Chain Builder', and 'Test Run'. Below this, a heading reads 'Enter parameters for each algorithm in the chain'. There are two input fields: 'API Key*' (empty) and 'My First Algorithm Name of person to greet*' (containing 'Chris'). A note states 'Fields marked with a * are required'. A blue button labeled 'Run Algorithm Chain' is positioned below the inputs. Below the button, the 'Chain Status' section shows two progress bars, both at 100%, labeled 'Algorithm Progress:' and 'Chain Progress:'. A text box below these bars contains the message 'Chain run complete'. At the bottom, the 'Chain Result:' section displays the output 'Hello, Chris!'.

This is fantastic. You can create algorithms that accept user input, process that input, and display something to the user. In all, you wrote six lines of code and filled out a couple of forms. Not bad.

1.7.8 Where To Go From Here?

Although this was a basic example, you've seen how most of the development environment works and how you can use the ATK to run an application. You're ready to take on a bigger project.

We suggest you go through the next section [Working With Chains](#) to understand how chains work and how to manipulate them. Then, you can run through the [Tutorial: Do Maths](#) to see how you can practically use processing chains. You'll also learn how to log information and provide status updates to the user.

1.8 Working With Chains

Now you've seen how to create a single algorithm and make it do something. Pretty cool, but the power of the ATK really shines with processing chains.

1.8.1 What Is A Processing Chain?

Simply put, a processing chain in the ATK is a series of algorithms, each linking to the next via output parameters.

Imagine a chain like a set of tasks. When you mail a package to someone, your task is complete when you drop it off at the shipping company. But to get to the recipient the package has to go through a series of waystations, each with its own tasks. The whole chain might look like this:

1. You drop off your package at the post office in the small town you live in
2. The post office sends it to a central office somewhere else in the state
3. The state hub sends the package to another hub
4. That hub sends the package to a local office near the recipient
5. A local delivery person drops it off at the recipient's location

At each step, information is read about where the package goes next. In addition, the final destination address has to be carried along the route so it ends up in the right place.

Think of the next destination along the route as an input to an algorithm, the journey between points as algorithms, and the information contained in the overall route as the Chain Ledger, and you've basically got the idea. After the package is delivered, you could do back and see all the steps the package went through, how long it took to reach each point, and so on.

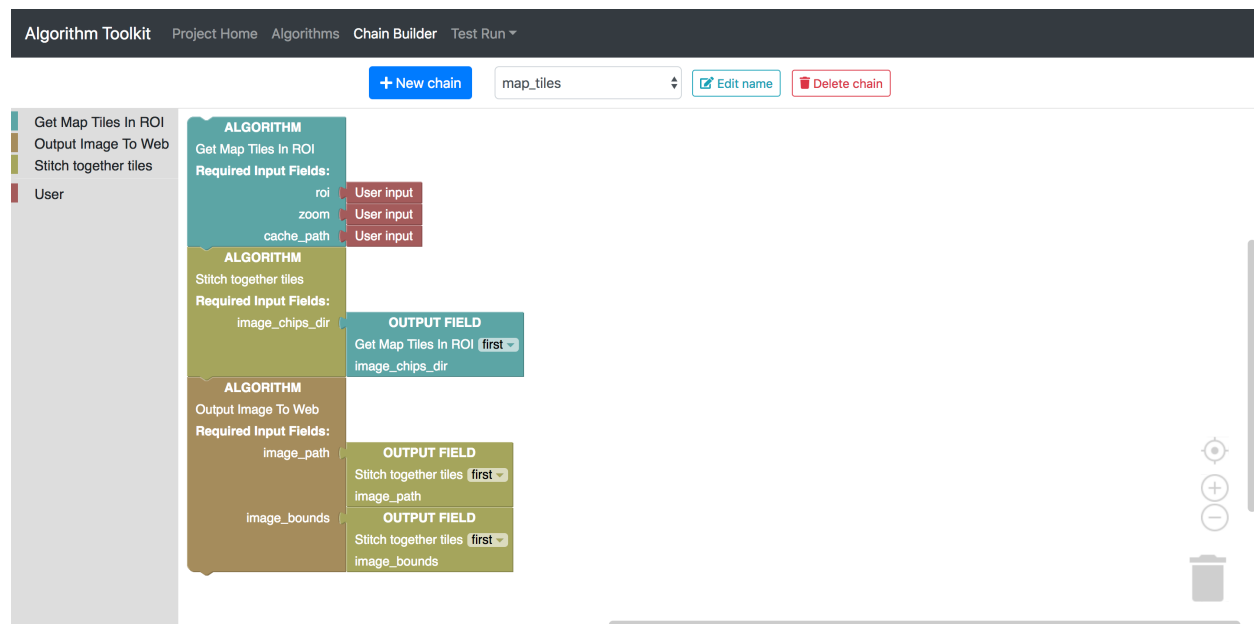
1.8.2 How Does This Actually Work?

You define algorithms just like you did in the last section *Creating Your First Algorithm*. Then, you link those algorithms together to create a workflow, or processing chain. Then you can test the chain using our test harness.

1.8.3 Examining The Example Project

Sometimes the best way to learn is to read working code. The example project comes with three algorithms and a processing chain; let's examine those.

Go to the Chain Builder in the web interface and select the “map_tiles” chain. This is what it looks like:



The left side of this screen contains the algorithms installed in your project. To the right of that and taking up most of the screen, is the Chain Builder Canvas. As you can see, “map_tiles” links the algorithms together in the following order (top to bottom):

1. Get Map Tiles in ROI
2. Stitch together tiles
3. Output Image to Web

You’ll also notice that each algorithm is getting its data from different sources. The three inputs for “Get Map Tiles in ROI” come from the user, but the other two algorithms get data from algorithms that came before. “Stitch together tiles” gets its one input value from an *output* of “Get Map Tiles in ROI” called “image_chips_dir”. “Stitch together tiles” also provides two outputs for “Output Image to Web”: “image_path” and “image_bounds”.

Let’s take a step back and talk about what this chain actually does. In the first algorithm, this chain acquires map tiles from a public tile server (provided by the U.S. Geological Survey) for a given region of interest (ROI). The chain then hands those tiles over to a second algorithm, which stitches the image tiles together into a GeoTIFF mosaic that is orthorectified. Finally, the chain hands the mosaic to a third algorithm that converts the GeoTIFF to a web format (PNG) and sends the PNG data along with the geographic bounding box data back to the client.

To use this chain, it’s not really necessary for you to know *how* the individual algorithms do what they do. All you need to know is what information goes in, and what comes out. However, because you have the source code for the algorithms in the chain, you can see how it all works.

That’s the beauty of the ATK platform: you can use it to do some really complicated things and have it “just work,” and you can also see the details about how the work was done.

We built the ATK specifically to have these two capabilities. Many who use the ATK rely on its power to make their data processing lives easier, while others need to tweak the algorithms in order to understand fully the results of a chain run.

Chain Definition Files

The Chain Builder provides a graphical view for a chain’s structure and lets you change how it’s put together. Under the hood, the chain is defined by JSON files in your project called `chains/<CHAIN_NAME>.json`. Here’s what one looks like with the example project:

```
{
  "map_tiles": [
    {
      "algorithm": "getmaptiles_roi",
      "parameter_source": "user"
    },
    {
      "parameters": {
        "image_chips_dir": {
          "source": "chain_ledger",
          "occurrence": "first",
          "key": "image_chips_dir",
          "source_algorithm": "getmaptiles_roi"
        }
      },
      "algorithm": "stitch_tiles"
    },
    {
      "parameters": {
        "image_bounds": {
```

(continues on next page)

(continued from previous page)

```

        "source": "chain_ledger",
        "occurrence": "first",
        "key": "image_bounds",
        "source_algorithm": "stitch_tiles"
    },
    "image_path": {
        "source": "chain_ledger",
        "occurrence": "first",
        "key": "image_path",
        "source_algorithm": "stitch_tiles"
    }
},
"algorithm": "output_image_to_client"
}
]
}

```

This JSON file would be called `chains/map_tiles.json`. You can see the same structure depicted in graphical form here:

- “getmaptiles_roi” gets its input from “user”
- “stitch_tiles” its one input from “getmaptiles_roi”
- “output_image_to_client” gets both of its inputs from “stitch_tiles”

If an algorithm gets all of its inputs from the user, you will see “parameter_source”: “user”. Otherwise, the next link in the processing chain will have a “parameters” dictionary applied, with each parameter named as a key: value pair. If the value comes from the user, it will read:

```

{
  "source": "user"
}

```

Otherwise, it will have the additional fields as shown here:

```

{
  "source": "chain_ledger",
  "occurrence": "first",
  "key": "",                                <-- name of parameter in source algorithm
  "source_algorithm": ""                   <-- name of the source algorithm
}

```

What is ‘occurrence’?

Some processing chains use the same algorithm more than once. In these cases, the “occurrence” flag is a mechanism to tell the ATK which occurrence of the algorithm to use for this input. The Chain Builder algorithm blocks have a small drop-down menu for the chain developer to use to indicate this value.

1.8.4 The Chain In Action

When you pull up a chain in the Test Run page, you will see its web form:

Algorithm Toolkit Project Home Algorithms Chain Builder Test Run ▾

Enter parameters for each algorithm in the chain

API Key*

Fields marked with a * are required

Get Map Tiles In ROI

Polygon WKT*

Enter well known text (WKT) for the polygon region to obtain intersecting tiles.

Zoom level*

Enter an integer corresponding to the zoom level, 16 is max value supported.

Tile Cache Location*

Absolute path to directory for saving tiles

Stitch together tiles

No input needed.

Output Image To Web

No input needed.

[Run Algorithm Chain](#)

What you see here is probably making more sense now. This view is presented by the ATK as an interface to the processing chain. In fact, the data to run a chain does not have to come from a web form at all; however, it's a convenient way to explain how this all works.

Copy and paste your API Key into the field provided. You'll notice that all the other fields are filled out: those are the default values defined by the algorithm creator. Especially in the case of the ROI field, this can really make a user's life easier. Let's keep these defaults and click "Run Algorithm Chain".

Algorithm Toolkit
Project Home
Algorithms
Chain Builder
Test Run

Enter parameters for each algorithm in the chain

API Key*

Fields marked with a * are required

Get Map Tiles In ROI

Polygon WKT*

POLYGON((-77.0419692993164 38.9933585922412,-77.17311859130861 38.891887936025896,-77

Enter well known text (WKT) for the polygon region to obtain intersecting tiles.

Zoom level*

14

Enter an integer corresponding to the zoom level, 16 is max value supported.

Tile Cache Location*

/tmp/tiles/map_tiles

Absolute path to directory for saving tiles

Stitch together tiles

No input needed.

Output Image To Web

No input needed.

Run Algorithm Chain

Chain Status:

Algorithm Progress:
100%

Chain Progress:
100%

Starting chain run...
Running algorithm: getmaptiles_roi
Running algorithm: stitch_tiles
Running algorithm: output_image_to_client
Chain run complete
Chain run complete

Chain Result:

When the chain runs, you'll see some status information and a progress bar indicating how far along the chain we are. An algorithm developer can also pass algorithm progress to the user as we'll see in another section.

In this case, the final output to the client appears in a web map. How did this happen?

Chain outputs

In the section *Creating Your First Algorithm*, you output text to the test client. The ATK allows you to output a variety of other formats:

geo_raster

Send a PNG or JPG to the client along with the geographic bounds (in lat,lon pairs) of the image. The image will display in a web map.

Format of chain_output_value:

```
{
  "output_type": "geo_raster",
  "output_value": {
    "extent": "",
    "raster": ""
  }
}
```

- extent is a nested array of lat,lon values (using the default ROI, the value would be: `[[38.99357205820944,-77.18994140625],[38.788345355085625,-76.90429687500001]]`)
- raster is a base64 encoded string of the image data

geojson

Send a GeoJSON object to the client. It will display in a web map.

Format of chain_output_value:

```
{
  "output_type": "geojson",
  "output_value": {}
}
```

- the output_value must be properly formatted GeoJSON

json

Send a JSON object to the client. It will display in a text box.

Format of chain_output_value:

```
{
  "output_type": "json",
  "output_value": {}
}
```

csv

Send a comma-delimited text file to the client. It will display in a text box.

Format of chain_output_value:

```
{
  "output_type": "csv",
  "output_value": {
    "data": ""
  }
}
```

- data should be a series of lines of comma-delimited text separated by newline characters

binary

Send a binary file to the user. The user will be prompted to download the file.

Format of chain_output_value:

```
{
  "output_type": "binary",
  "output_value": {
    "mimetype": "",
    "file": "",
    "filename": ""
  }
}
```

- mimetype is the MIME encoding for the file; if not provided, the file will be rejected by the test client (e.g.: “application/pdf”)
- file is a base64 encoded string of the binary file
- filename is the name of the file

text

Send a string to the client.

Format of chain_output_value:

```
{
  "output_type": "text",
  "output_value": ""
}
```

1.9 The Chain Ledger

The Chain Ledger is a powerful feature of the ATK. It lets you store arbitrary data throughout the course of a chain run. By default the Chain Ledger will contain all the inputs provided to each algorithm in the chain.

In your algorithms, you can store and retrieve values from the Ledger easily by using the `cl` object in your algorithm’s `main.py` (see [Working With Algorithms](#)). Here’s how:

```
# store a value
cl.add_to_metadata('my key', 'my value')

# retrieve a value stored by your algorithm
cl.get_from_metadata('my key')
```

1.9.1 Chain History

Throughout its life, your processing chain stores values on the Ledger. Often, you’ll want to retrieve a value stored by an earlier algorithm in the chain. If you want a value from the algorithm that immediately preceded the current one, typically you would use an **output** and add that output to the first algorithm’s definition (again, see [Working With Algorithms](#)). Then you would map the output to the second algorithm’s input where appropriate (see [Working With Chains](#)).

(continued from previous page)

```

    },
    {
        "algorithm_name": "stitch_tiles",
        "algorithm_params": {
            "image_filenames": "/tmp/flhp6CrpMadJUxmNEi/temp/4684_6273_14.png,/
→tmp/flhp6CrpMadJUxmNEi/temp/4685_6273_14.png,/tmp/flhp6CrpMadJUxmNEi/temp/4686_6273_
→14.png,/tmp/flhp6CrpMadJUxmNEi/temp/4683_6272_14.png,/tmp/flhp6CrpMadJUxmNEi/temp/
→4684_6272_14.png,/tmp/flhp6CrpMadJUxmNEi/temp/4685_6272_14.png,/tmp/
→flhp6CrpMadJUxmNEi/temp/4686_6272_14.png,/tmp/flhp6CrpMadJUxmNEi/temp/4687_6272_14.
→png,/tmp/flhp6CrpMadJUxmNEi/temp/4682_6271_14.png,/tmp/flhp6CrpMadJUxmNEi/temp/4683_
→6271_14.png,/tmp/flhp6CrpMadJUxmNEi/temp/4684_6271_14.png,/tmp/flhp6CrpMadJUxmNEi/
→temp/4685_6271_14.png,/tmp/flhp6CrpMadJUxmNEi/temp/4686_6271_14.png,/tmp/
→flhp6CrpMadJUxmNEi/temp/4687_6271_14.png,/tmp/flhp6CrpMadJUxmNEi/temp/4688_6271_14.
→png,/tmp/flhp6CrpMadJUxmNEi/temp/4681_6270_14.png,/tmp/flhp6CrpMadJUxmNEi/temp/4682_
→6270_14.png,/tmp/flhp6CrpMadJUxmNEi/temp/4683_6270_14.png,/tmp/flhp6CrpMadJUxmNEi/
→temp/4684_6270_14.png,/tmp/flhp6CrpMadJUxmNEi/temp/4685_6270_14.png,/tmp/
→flhp6CrpMadJUxmNEi/temp/4686_6270_14.png,/tmp/flhp6CrpMadJUxmNEi/temp/4687_6270_14.
→png,/tmp/flhp6CrpMadJUxmNEi/temp/4688_6270_14.png,/tmp/flhp6CrpMadJUxmNEi/temp/4689_
→6270_14.png,/tmp/flhp6CrpMadJUxmNEi/temp/4680_6269_14.png,/tmp/flhp6CrpMadJUxmNEi/
→temp/4681_6269_14.png,/tmp/flhp6CrpMadJUxmNEi/temp/4682_6269_14.png,/tmp/
→flhp6CrpMadJUxmNEi/temp/4683_6269_14.png,/tmp/flhp6CrpMadJUxmNEi/temp/4684_6269_14.
→png,/tmp/flhp6CrpMadJUxmNEi/temp/4685_6269_14.png,/tmp/flhp6CrpMadJUxmNEi/temp/4686_
→6269_14.png,/tmp/flhp6CrpMadJUxmNEi/temp/4687_6269_14.png,/tmp/flhp6CrpMadJUxmNEi/
→temp/4688_6269_14.png,/tmp/flhp6CrpMadJUxmNEi/temp/4689_6269_14.png,/tmp/
→flhp6CrpMadJUxmNEi/temp/4690_6269_14.png,/tmp/flhp6CrpMadJUxmNEi/temp/4679_6268_14.
→png,/tmp/flhp6CrpMadJUxmNEi/temp/4680_6268_14.png,/tmp/flhp6CrpMadJUxmNEi/temp/4681_
→6268_14.png,/tmp/flhp6CrpMadJUxmNEi/temp/4682_6268_14.png,/tmp/flhp6CrpMadJUxmNEi/
→temp/4683_6268_14.png,/tmp/flhp6CrpMadJUxmNEi/temp/4684_6268_14.png,/tmp/
→flhp6CrpMadJUxmNEi/temp/4685_6268_14.png,/tmp/flhp6CrpMadJUxmNEi/temp/4686_6268_14.
→png,/tmp/flhp6CrpMadJUxmNEi/temp/4687_6268_14.png,/tmp/flhp6CrpMadJUxmNEi/temp/4688_
→6268_14.png,/tmp/flhp6CrpMadJUxmNEi/temp/4689_6268_14.png,/tmp/flhp6CrpMadJUxmNEi/
→temp/4690_6268_14.png,/tmp/flhp6CrpMadJUxmNEi/temp/4691_6268_14.png,/tmp/
→flhp6CrpMadJUxmNEi/temp/4679_6267_14.png,/tmp/flhp6CrpMadJUxmNEi/temp/4680_6267_14.
→png,/tmp/flhp6CrpMadJUxmNEi/temp/4681_6267_14.png,/tmp/flhp6CrpMadJUxmNEi/temp/4682_
→6267_14.png,/tmp/flhp6CrpMadJUxmNEi/temp/4683_6267_14.png,/tmp/flhp6CrpMadJUxmNEi/
→temp/4684_6267_14.png,/tmp/flhp6CrpMadJUxmNEi/temp/4685_6267_14.png,/tmp/
→flhp6CrpMadJUxmNEi/temp/4686_6267_14.png,/tmp/flhp6CrpMadJUxmNEi/temp/4687_6267_14.
→png,/tmp/flhp6CrpMadJUxmNEi/temp/4688_6267_14.png,/tmp/flhp6CrpMadJUxmNEi/temp/4689_
→6267_14.png,/tmp/flhp6CrpMadJUxmNEi/temp/4690_6267_14.png,/tmp/flhp6CrpMadJUxmNEi/
→temp/4691_6267_14.png,/tmp/flhp6CrpMadJUxmNEi/temp/4680_6266_14.png,/tmp/
→flhp6CrpMadJUxmNEi/temp/4681_6266_14.png,/tmp/flhp6CrpMadJUxmNEi/temp/4682_6266_14.
→png,/tmp/flhp6CrpMadJUxmNEi/temp/4683_6266_14.png,/tmp/flhp6CrpMadJUxmNEi/temp/4684_
→6266_14.png,/tmp/flhp6CrpMadJUxmNEi/temp/4685_6266_14.png,/tmp/flhp6CrpMadJUxmNEi/
→temp/4686_6266_14.png,/tmp/flhp6CrpMadJUxmNEi/temp/4687_6266_14.png,/tmp/
→flhp6CrpMadJUxmNEi/temp/4688_6266_14.png,/tmp/flhp6CrpMadJUxmNEi/temp/4689_6266_14.
→png,/tmp/flhp6CrpMadJUxmNEi/temp/4690_6266_14.png,/tmp/flhp6CrpMadJUxmNEi/temp/4681_
→6265_14.png,/tmp/flhp6CrpMadJUxmNEi/temp/4682_6265_14.png,/tmp/flhp6CrpMadJUxmNEi/
→temp/4683_6265_14.png,/tmp/flhp6CrpMadJUxmNEi/temp/4684_6265_14.png,/tmp/
→flhp6CrpMadJUxmNEi/temp/4685_6265_14.png,/tmp/flhp6CrpMadJUxmNEi/temp/4686_6265_14.
→png,/tmp/flhp6CrpMadJUxmNEi/temp/4687_6265_14.png,/tmp/flhp6CrpMadJUxmNEi/temp/4688_
→6265_14.png,/tmp/flhp6CrpMadJUxmNEi/temp/4689_6265_14.png,/tmp/flhp6CrpMadJUxmNEi/
→temp/4682_6264_14.png,/tmp/flhp6CrpMadJUxmNEi/temp/4683_6264_14.png,/tmp/
→flhp6CrpMadJUxmNEi/temp/4684_6264_14.png,/tmp/flhp6CrpMadJUxmNEi/temp/4685_6264_14.
→png,/tmp/flhp6CrpMadJUxmNEi/temp/4686_6264_14.png,/tmp/flhp6CrpMadJUxmNEi/temp/4687_
→6264_14.png,/tmp/flhp6CrpMadJUxmNEi/temp/4688_6264_14.png,/tmp/flhp6CrpMadJUxmNEi/
→temp/4683_6263_14.png,/tmp/flhp6CrpMadJUxmNEi/temp/4684_6263_14.png,/tmp/
→flhp6CrpMadJUxmNEi/temp/4685_6263_14.png,/tmp/flhp6CrpMadJUxmNEi/temp/4686_6263_14.
→png,/tmp/flhp6CrpMadJUxmNEi/temp/4687_6263_14.png,/tmp/flhp6CrpMadJUxmNEi/temp/4688_
→6262_14.png,/tmp/flhp6CrpMadJUxmNEi/temp/4685_6262_14.png,/tmp/flhp6CrpMadJUxmNEi/
→temp/4686_6262_14.png"

```

(continues on next page)

(continued from previous page)

```

    },
    "image_path": "/tmp/flhp6CrpMadJUxmNEi/temp/stitched_tiles.png",
    "image_bounds": "[[38.99357205820944, -77.18994140625], [38.
↪788345355085625, -76.90429687500001]]"
  },
  {
    "algorithm_name": "output_image_to_client",
    "image_extent": "[[38.99357205820944, -77.18994140625], [38.
↪788345355085625, -76.90429687500001]]",
    "algorithm_params": {
      "image_bounds": "[[38.99357205820944, -77.18994140625], [38.
↪788345355085625, -76.90429687500001]]",
      "image_path": "/tmp/flhp6CrpMadJUxmNEi/temp/stitched_tiles.png"
    },
    "image_url": "/tmp/flhp6CrpMadJUxmNEi/temp/stitched_tiles.png"
  }
]
}

```

As you can see, even a relatively simple algorithm can generate a lot of data! This is extremely helpful though, because it lets you see exactly what happened over the course of the chain run.

1.9.2 Best Practices

Make good use of the Chain Ledger, not just because it helps you keep track of values throughout a chain run but also because it helps you reproduce results at a later time. The history will also help other people who use your algorithms in their chains.

1.10 Tutorial: Do Maths

OK, with your Hello World algorithm under your belt, it's time to see more of what the ATK can really do. In this tutorial, you'll learn the following:

- How to chain multiple algorithms together
- How to log messages to a file from your algorithms
- How to provide status updates to the user
- How to use the Chain Ledger
- How to test your algorithm

We're calling this "Do Maths" because that's what the chain will do for us. We'll create a series of algorithms that together calculate a value, and present that value to the user. This tutorial assumes you went through the Hello World example earlier, so we won't go into detail for areas covered in that section.

Let's dive right in!

1.10.1 Setting Up

First, create a project. You can use the same one you created for *Creating Your First Algorithm*.

Next, we'll create an algorithm called "add_numbers". It should have two required parameters and one output:

Parameters:

- `starting_value`: an integer
- `number_to_add`: another integer

Output:

- `result`: yet another integer

If you went through the *Creating Your First Algorithm* section, this should be a breeze. Your algorithm form should look like this:

Algorithm Toolkit

Project Home

Algorithms

Chain Builder

Documentation

Test Run

Create/Edit an Algorithm

Algorithm Name

add_numbers

Enter a short name for the algorithm (no spaces)

Display Name

Add two numbers together

Enter a more descriptive name for the algorithm

Description

Add two numbers together to get a result.

Version

0.0.1

Enter a version string (e.g.: "0.0.1", "beta")

Algorithm Website

Include a URL where someone could go for documentation

Private

☐

Make this algorithm unlisted (if publishing to the Algorithm Registry)

License

MIT

See <https://choosealicense.com/licenses/> for info on various open source licenses

Algorithm Input Parameters:

Add a parameter

Name	Description	Required
starting_value	The starting value for the add operation.	Yes
number_to_add	The number to add to the starting value.	Yes

Algorithm Outputs:

Add an output

Name	Description
result	The result of the add operation.

Save Algorithm

Cancel

☐ Also update README file

© You 2019

1.10.2 The First Algorithm

Save your new algorithm, copy the code below and paste into the algorithm's `main.py`:

```
import time
from algorithm_toolkit import Algorithm, AlgorithmChain

class Main(Algorithm):
```

(continues on next page)

(continued from previous page)

```

def run(self):
    cl = self.cl # type: AlgorithmChain
    params = self.params # type: dict
    # Add your algorithm code here

    starting_value = params['starting_value']
    number_to_add = params['number_to_add']

    result = starting_value + number_to_add

    cl.add_to_metadata('result', result)

    status_msg = 'result so far: ' + str(result)
    cl.set_status(status_msg)

    chain_output = {
        'output_type': 'text',
        'output_value': status_msg
    }
    cl.add_to_metadata('chain_output_value', chain_output)
    time.sleep(1)

    # Do not edit below this line
    return cl

```

We’ve added a bunch of code to this algorithm. Let’s go through it step by step:

```
import time
```

We’re adding a delay for testing purposes so that you can see the chain status messages appear. Otherwise, this chain would run too quickly.

```
starting_value = params['starting_value']
number_to_add = params['number_to_add']
```

Here we’re creating local variables for parameters coming in from the ATK. Because we specified Integer data types, we can be sure that these are integers.

```
result = starting_value + number_to_add
```

This line simply adds the two values together, which is the primary purpose of this algorithm.

```
cl.add_to_metadata('result', result)
```

This places the result of the add operation onto the Chain Ledger. Note that we used the key “result”. This is important, because it’s the name of the output we defined for this algorithm. That means that another algorithm that comes after this one can receive and use this value (which we said we would provide in the algorithm definition).

The other important point here is that we’re placing the value onto the Chain Ledger as an integer. Since we specified that the “result” output would be of data type Integer, we have to keep our word.

```
status_msg = 'result so far: ' + str(result)
cl.set_status(status_msg)
```

These two lines create a status message we will send to the client. Using the `set_status()` function of the Chain Ledger places our status message into a special global variable that can be retrieved by other parts of the ATK. When

we run the chain, you'll see this in action.

```
chain_output = {
    'output_type': 'text',
    'output_value': status_msg
}
cl.add_to_metadata('chain_output_value', chain_output)
```

This should look familiar. We're outputting the same status message to `chain_output_value` in case this is the last algorithm in the chain.

Note: You should always output something to the chain's `chain_output_value` in case your algorithm is last in a chain. It's not a requirement, but it's a good practice.

```
time.sleep(1)
```

This is the delay timer. Each algorithm will wait one second before completing.

That's it! We then return the modified Chain Ledger (`cl`) to the ATK for it to pass to the next algorithm in the chain.

1.10.3 Testing Your Algorithm

Although we said earlier that the ATK doesn't run individual algorithms, we can write code to test each algorithm. If you've written tests for Python programs before, then you're in luck: the ATK uses Python's built-in testing tools.

Our algorithm has basically only one thing we need to test: does it add two numbers together correctly? To test this, we'll make use of some functionality included in the ATK.

The test module

When you create a new algorithm, the ATK will create a `test.py` module in the algorithm's folder. It will look like this:

```
from algorithm_toolkit import AlgorithmTestCase

from main import Main

class MainTestCase(AlgorithmTestCase):

    def runTest(self):
        # configure params for your algorithm
        self.params = {}

        self.alg = Main(cl=self.cl, params=self.params)
        self.alg.run()

        # Add tests and assertions below
```

Let's see what's happening here.

```
from algorithm_toolkit import AlgorithmTestCase
```

We're making use of a class that the ATK makes available to us: `AlgorithmTestCase`. This class sets up our tests and provides two attributes and two helper functions:

- `cl`: a Chain Ledger object for testing purposes
- `params`: an empty dictionary you can use to “fake” inputs to your algorithm
- `check_metadata()`: this ensures that the value we expect shows up on the Chain Ledger
- `check_status()`: this ensures that we are writing status information correctly

```
from main import Main
```

Here we’re simply importing the `Main` class in our algorithm’s `main.py` module.

```
self.alg = Main(cl=self.cl, params=self.params)
```

This looks complicated but it really isn’t. All this code does is call your algorithm’s `Main` class and pass in the same `cl` and `params` attributes you’re used to seeing in the algorithm code.

```
self.alg.run()
```

This calls your algorithm’s `run()` function.

With this structure, you can test anything you want about your algorithm. To test “add_numbers”, we’ll need some numbers. Modify the `self.params = {}` line as follows:

```
self.params = {
    'starting_value': 3,
    'number_to_add': 5
}
```

Here we “fake” our algorithm’s inputs. We’ll make sure “add_numbers” can add $3 + 5$ and get the right result.

Next, add these two assertions:

```
self.assertTrue(self.check_metadata('result', 8))
self.assertTrue(self.check_status('result so far: 8'))
```

We’re making use of the two helper functions mentioned above. Here we make sure that the number 8 is added to a key called “result” on the Chain Ledger. We also make sure that we’re writing the correct status message for the user.

The whole `test.py` should look like this:

```
from algorithm_toolkit import AlgorithmTestCase

from main import Main

class AddNumbersTestCase(AlgorithmTestCase):

    def runTest(self):
        # configure params for your algorithm
        self.params = {
            'starting_value': 3,
            'number_to_add': 5
        }

        self.alg = Main(cl=self.cl, params=self.params)
        self.alg.run()

        # Add tests and assertions below
```

(continues on next page)

(continued from previous page)

```
self.assertTrue(self.check_metadata('result', 8))
self.assertTrue(self.check_status('result so far: 8'))
```

Running the test

Stop the development environment by typing `^C` in your Terminal. Use the CLI `test` command to run your tests:

```
alg test
```

Simple, isn't it? Note that this command will check all of the algorithms in your project and run any tests it finds. If you only want to test one algorithm, just include the algorithm's name:

```
alg test add_numbers
```

After you run the test, you should see a message like this in your Terminal window:

```
runTest (test.AddNumbersTestCase) ... result so far: 8
ok

-----
Ran 1 test in 0.001s

OK
```

Not very exciting, but good news: our test passed.

Adding more tests

Notice that the message says “Ran 1 test”. This may seem odd, since we made two assertions. The way Python's unittest works, the test function (`runTest()` in this case) counts as one test no matter how many assertions you make.

Some developers like to split things out so that each test only makes one assertion. You can easily add more tests by changing things up a bit. Let's split our assertions into a test called `test_metadata()` and one called `test_status()`:

Note: When you have a single test in a Python `TestCase`, you typically override the `runTest()` function. When you have multiple tests, you name each function `test_` and add the name of the test. See Python's [documentation](#) on `unittest` for more info.

Here's our new `test.py`:

```
from algorithm_toolkit import AlgorithmTestCase

from main import Main

class AddNumbersTestCase(AlgorithmTestCase):

    def runTest(self):
        self.alg = Main(cl=self.cl, params=self.params)
        self.alg.run()
```

(continues on next page)

(continued from previous page)

```

def test_metadata(self):
    # configure params for your algorithm
    self.params = {
        'starting_value': 3,
        'number_to_add': 5
    }
    self.runTest()

    # Add tests and assertions below

    self.assertTrue(self.check_metadata('result', 8))

def test_status(self):
    # configure params for your algorithm
    self.params = {
        'starting_value': 17,
        'number_to_add': 23
    }
    self.runTest()

    # Add tests and assertions below

    self.assertTrue(self.check_status('result so far: 40'))

```

We keep the `runTest()` function and give it the job of calling the `Main` class and `run()` function from our algorithm. This helps eliminate some redundancy between tests. Notice that we're also changing up the input parameters just to make sure everything's working. Use `alg test` again and you should see two tests:

```

test_metadata (test.AddNumbersTestCase) ... ok
test_status (test.AddNumbersTestCase) ... ok

```

```

-----
Ran 2 tests in 2.006s

```

Now you can add as many tests as you like.

1.10.4 The Remaining Algorithms

We need to create three more algorithms that are basically copies of this one with a few tweaks. If your development environment is stopped, just start it again with:

```
alg run
```

Copy algorithm

Here's a handy feature. From the Algorithms page in the web interface, click the "Copy" button next to the "add_numbers" algorithm. When you do, you'll see this:

Algorithms in this project

[Create a new algorithm](#)

add_numbers	Edit	Copy	Delete
add_numbers_copy	Edit	Copy	Delete
getmaptiles_roi	Edit	Copy	Delete
my_first_algorithm	Edit	Copy	Delete
output_image_to_client	Edit	Copy	Delete
stitch_tiles	Edit	Copy	Delete

© You 2019

An exact duplicate of the “add_numbers” algorithm is now in your list, with a new name (“add_numbers_copy”). Now you can just make changes to that algorithm instead of creating one from scratch.

Click the “Edit” button next to the new algorithm. Change it’s name to “subtract_numbers”. Also change the display name and description, but keep everything else the same.

Modify subtract_numbers

You really just need to change two lines in the code from “add_numbers”. It should look like this when you’re done:

```
import time
from algorithm_toolkit import Algorithm, AlgorithmChain

class Main(Algorithm):

    def run(self):
        cl = self.cl # type: AlgorithmChain
        params = self.params # type: dict
        # Add your algorithm code here

        starting_value = params['starting_value']
        number_to_subtract = params['number_to_subtract']

        result = starting_value - number_to_subtract

        cl.add_to_metadata('result', result)

        status_msg = 'result so far: ' + str(result)
        cl.set_status(status_msg)

        chain_output = {
            'output_type': 'text',
```

(continues on next page)

(continued from previous page)

```

        'output_value': status_msg
    }
    cl.add_to_metadata('chain_output_value', chain_output)
    time.sleep(1)

    # Do not edit below this line
    return cl

```

Add two more algorithms

Now you'll create two more algorithms: “multiply_numbers” and “divide_numbers” by copying either “add_numbers” or “subtract_numbers”. After you're done your Algorithms page will look like this:

Algorithm Toolkit
Project Home
Algorithms
Chain Builder
Test Run

Algorithms in this project

Create a new algorithm

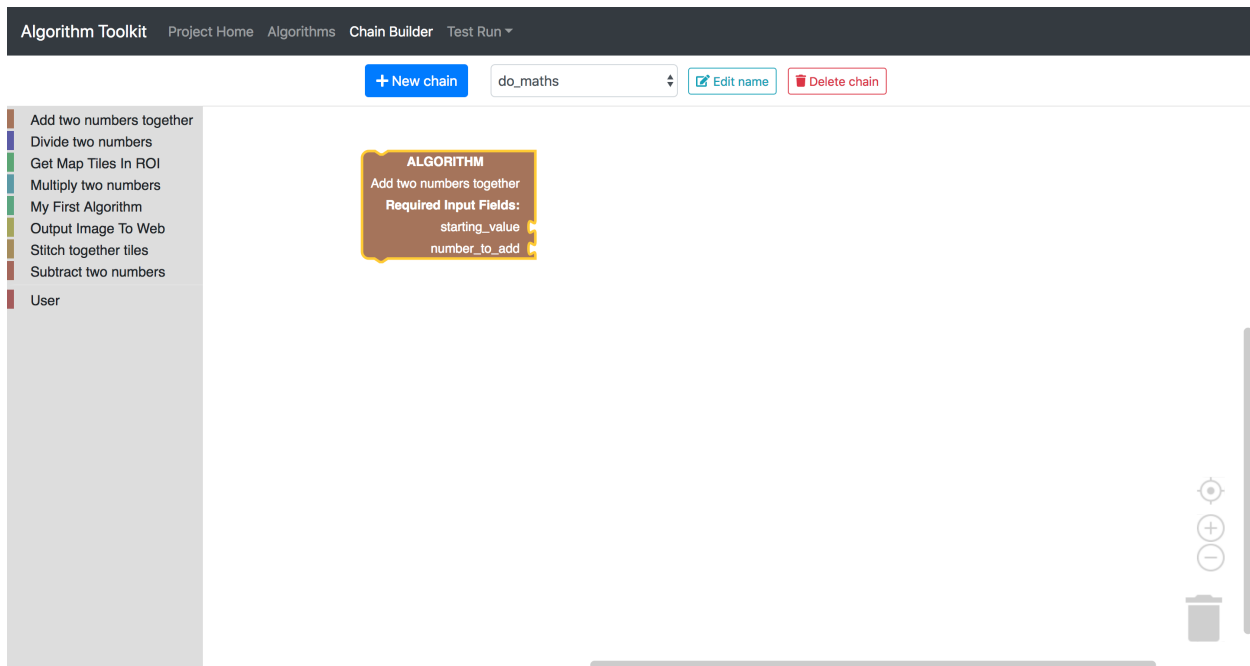
add_numbers	Edit	Copy	Delete
divide_numbers	Edit	Copy	Delete
getmaptiles_roi	Edit	Copy	Delete
multiply_numbers	Edit	Copy	Delete
my_first_algorithm	Edit	Copy	Delete
output_image_to_client	Edit	Copy	Delete
stitch_tiles	Edit	Copy	Delete
subtract_numbers	Edit	Copy	Delete

© You 2019

Make sure to modify these two algorithms' `main.py` files in the same way you changed “subtract_numbers”.

1.10.5 Creating The Chain

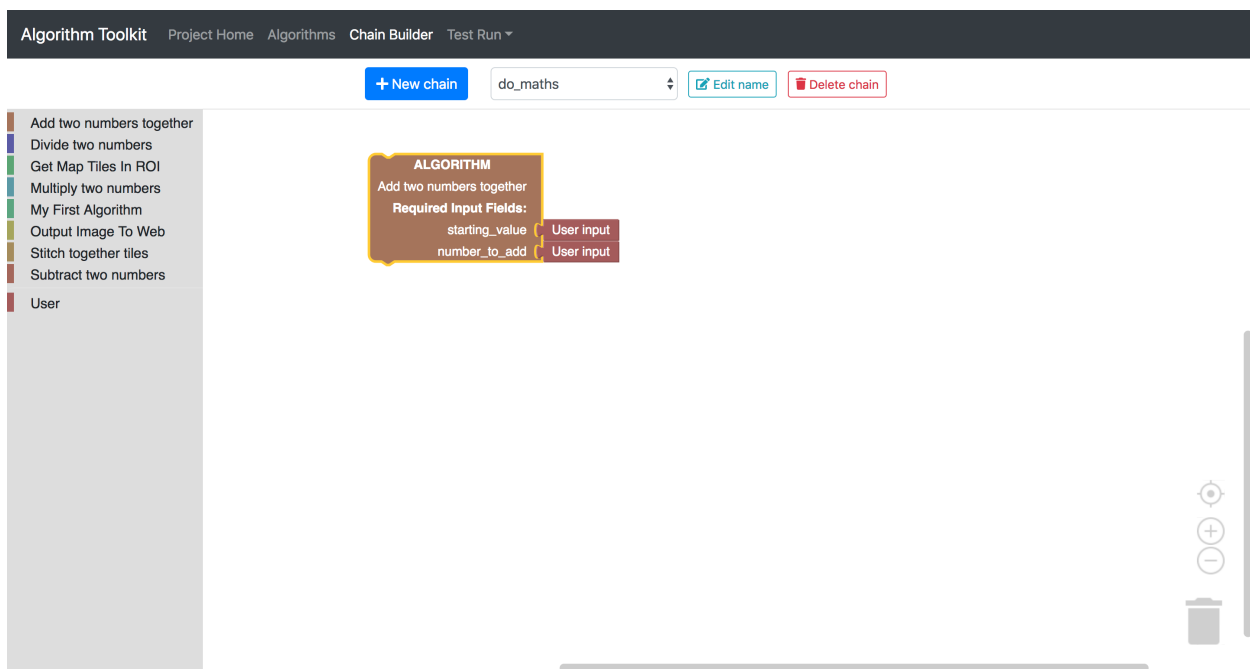
Now go to the Chain Builder so we can create our `do_maths` chain. Select “Add two numbers together” (or whatever display name you gave to the “add_numbers” algorithm). Drag its block into the Canvas:



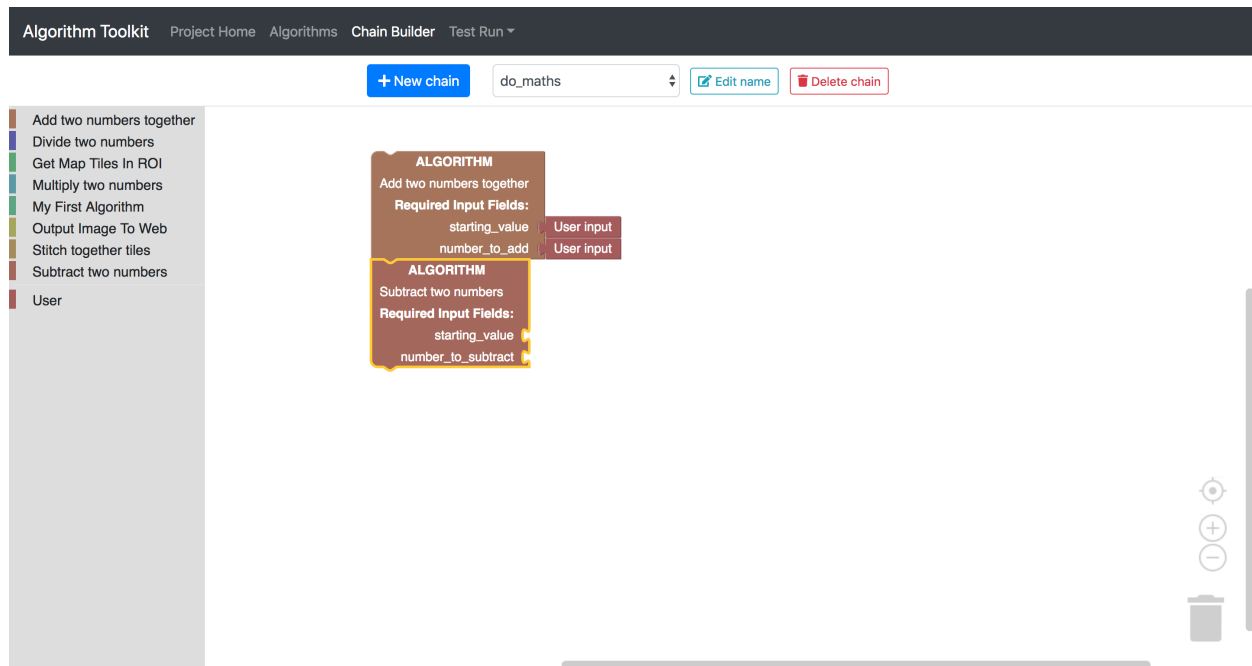
When the pop-up message appears asking for a chain name, call it “do_maths”.

Notice that when you clicked the “add_numbers” algorithm, in addition to the Algorithm block there was an Output Field block as well. This is important to remember, and we’ll be using it soon.

You now need to tell the ATK where the values for this new algorithm will come from. In this case, they are both User Inputs. You can either drag the User Input block into the two inputs, or highlight the “add_numbers” block in the canvas and hit the letter “u” on your keyboard. When you’re done, you should have this:



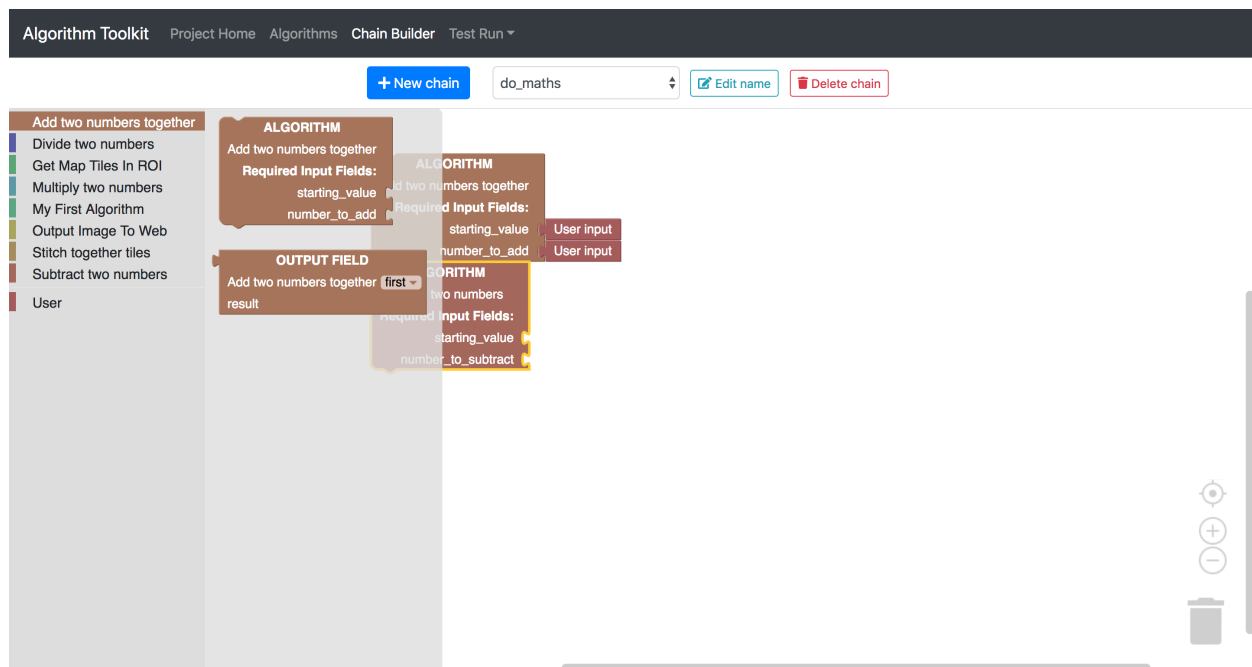
Now, drag “subtract_numbers” onto the canvas and attach it to “add_numbers”. You’ll hear a slight click when the two come together. The Canvas should look like this:



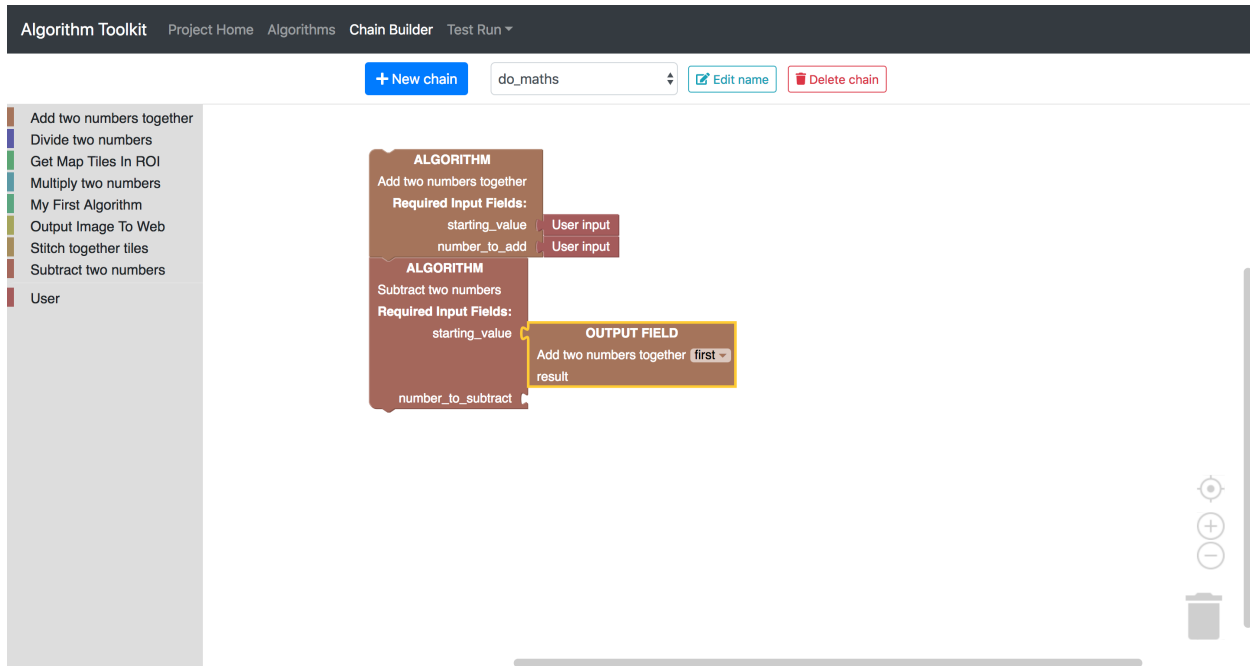
Linking an output to an input

Now here's the really critical part. The “starting_value” for this second algorithm should come from the result of the first algorithm's addition operation. How do we indicate that?

Remember the Output Field block from earlier? We'll use it here. Click the “add_numbers” algorithm on the left-side list. You'll see the Output Field block like so:



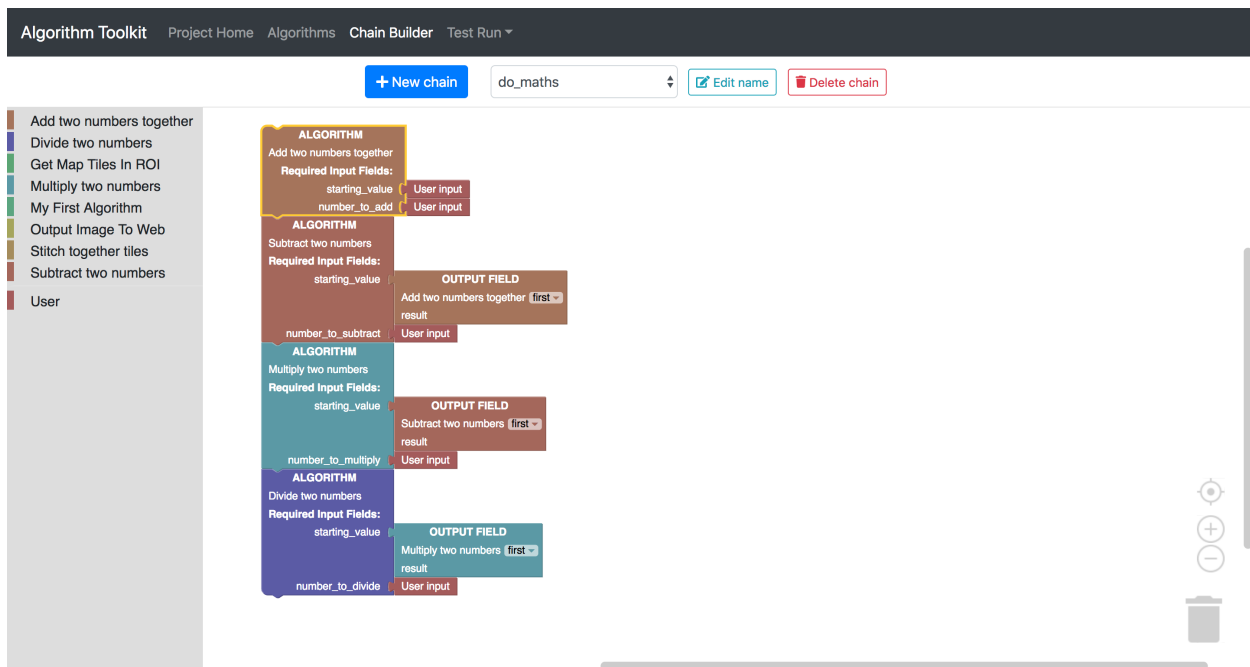
Now, drag the Output Field block and attach it to the “starting_value” input of “subtract_numbers”. The Canvas should now look like this:



Next, the “number_to_subtract” input should come from the user, so drag the User Input over and connect it to that input.

Continue building the chain this way, bringing in “multiply_numbers” and then “divide_numbers”. In each case, the “starting_value” should come from the algorithm that preceded it and the “number_to_multiply” or “number_to_divide” should come from the user.

When you’re done, your chain should look like this:



1.10.6 Running The Chain

Now let's try it out! Select the “do_maths” chain from the Test Run menu. You'll see our new web form:

[Algorithm Toolkit](#) [Project Home](#) [Algorithms](#) [Chain Builder](#) [Test Run ▾](#)

Enter parameters for each algorithm in the chain

API Key*

Fields marked with a * are required

Add two numbers together

Starting Value*

i

Number to Add*

i

Subtract two numbers

Number to Subtract*

i

Multiply two numbers

Number to Multiply*

i

Divide two numbers

Number to Divide*

i

Run Algorithm Chain

Notice that only the first algorithm in the chain (“add_numbers”) has a Starting Value field. That's because the Starting Value for the other algorithms come from the Chain Ledger, not the user.

Enter some numbers (and your API Key) and click the “Run Algorithm Chain” button. You should see the status messages appear in the status window, and the progress bars move each second. The result of the calculation will depend on what numbers you entered.

1.10. Tutorial: Do Maths

63

Algorithm Toolkit
Project Home
Algorithms
Chain Builder
Test Run

Enter parameters for each algorithm in the chain

API Key*

Fields marked with a * are required

Add two numbers together

Starting Value* 100 ⓘ

Number to Add* 60 ⓘ

Subtract two numbers

Number to Subtract* 10 ⓘ

Multiply two numbers

Number to Multiply* 3 ⓘ

Divide two numbers

Number to Divide* 5 ⓘ

Run Algorithm Chain

Chain Status:

Algorithm Progress: 100%

Chain Progress: 100%

```

Running algorithm: subtract_numbers
result so far: 150
Running algorithm: multiply_numbers
result so far: 450
Running algorithm: divide_numbers
result so far: 90
Chain run complete

```

Chain Result:

result so far: 90

1.10.7 Logging Information

Sometimes as a developer you want messages to be logged when there are problems, or just to keep a record of the status of your program. The ATK provides functionality to help you do this.

Let's say you want to record a log message after the last algorithm in the chain runs, and include the result of the maths operation. This is provided to the user at runtime, but maybe you want to save it for later.

Your algorithm project includes a folder called **logs**, and in that folder is a file called `app.log`. If you've run any chains so far, you will see some informational messages in there already telling you how long your chains took to run. We want to use this same log file to record our maths results.

Adding a log message is easy. Your algorithm has an attribute called `logger`, which you can use like so:

```
self.logger.info('My info message')
```

As you can see, we're passing a string to the log that reads "My info message". We're also calling it an "info" message by using `self.logger.info()`. This part of the command is important: it refers to the `LogLevel` used by Python's logging library. There are essentially five log levels in order from most to least severe:

1. CRITICAL
2. ERROR
3. WARNING

4. INFO
5. DEBUG

When you write a message to the log file using the corresponding function (e.g.: `info()`), the log level is checked against the current Log Handler’s log level. By default, the log handler used by the ATK is set to `DEBUG`. This means that any message as or more severe than `DEBUG` will be written to the log file.

If you raise the handler’s `LogLevel` to something else (say `ERROR`), then an `INFO` message will not appear in the logs. See [Algorithm Projects](#) for more information on configuring your project.

How this looks in your algorithm

To write the final result of your maths operation to the log file, you can do this in “divide_numbers”:

```
import time
from algorithm_toolkit import Algorithm

class Main(Algorithm):

    def run(self):
        cl = self.cl
        params = self.params
        # Add your algorithm code here

        starting_value = params['starting_value']
        number_to_divide = params['number_to_divide']

        result = starting_value / number_to_divide

        cl.add_to_metadata('result', result)

        status_msg = 'result so far: ' + str(result)
        cl.set_status(status_msg)

        chain_output = {
            'output_type': 'text',
            'output_value': status_msg
        }
        cl.add_to_metadata('chain_output_value', chain_output)
        time.sleep(1)

        self.logger.info('FINAL RESULT: ' + str(result)) # write the final result to
↳the log

        # Do not edit below this line
        return cl
```

Run the chain again and you’ll see your new log message in `app.log`. Experiment with other log messages and levels to understand how this simple but powerful feature works.

1.10.8 Next Steps

There’s lots you could add to this project. Here are a few things to figure out on your own:

- Make some or all of the algorithm input parameters float values instead of integers

- Rearrange the algorithms to perform the maths operation in a different order
- Use an algorithm more than once in the chain
- Make the maths more complicated (factorials? cube roots? trigonometric functions?)

Hopefully you can see the potential in the ATK. Without much code, you successfully ran a set of tasks and could be confident that the inputs and outputs would be present and what you expected to get.

The rest of these docs have some in-depth information about how the ATK works, and will introduce you to some other tools and platforms to enhance the ATK and make it even more useful.

1.11 CLI

The ATK includes a Command Line Interface (CLI) that helps you with a number of tasks. Several sections of these docs show you practical examples; this page contains all the commands and options for each.

1.11.1 alg

All of the commands are prefaced by the word `alg`.

Syntax:

```
Usage: alg [OPTIONS] COMMAND [ARGS]...
```

If you type `alg` by itself, you'll get the help message:

```
Usage: alg [OPTIONS] COMMAND [ARGS]...

Welcome to the Algorithm Toolkit CLI!

Options:
  --help  Show this message and exit.

Commands:
  ca          Create an algorithm.
  cp          Create an algorithm project.
  generate_settings  Generate new environment variables for your...
  info        Get detailed info about an algorithm from the...
  install     Install an algorithm from the Registry.
  list        List algorithms in the current project.
  publish     Publish an algorithm to the Algorithm...
  run
  search      Search the Algorithm Registry for algorithm...
  shell
  test        Test algorithms in your project.
  uninstall   Remove an algorithm from your project.
```

The `--help` or `-h` option will also display this message. Using `--help` with a command will display help for that command; e.g.:

```
alg cp --help
```

1.11.2 Commands

cp

Create a new algorithm project.

Syntax:

```
Usage: alg cp [OPTIONS] PROJECT_NAME

    Create an algorithm project.

Options:
  -e, --example      Install example project
  -wd, --with-docs   Add documentation to portal
  -q, --quiet        Suppress screen output
  -h, --help         Show this message and exit.
```

Use the `-e` option to install the example project.

Use the `-wd` option to install local documentation (same as these docs).

Use the `-q` option to suppress output messages from this command.

ca

Create a new algorithm.

Syntax:

```
Usage: alg ca [OPTIONS] ALGORITHM_NAME

    Create an algorithm.

Options:
  -h, --help   Show this message and exit.
```

When you create a new algorithm, the CLI steps you through each of the algorithm fields of information one at a time. You can also create parameters and outputs by responding to additional questions.

generate_settings

Create a new `.env` file for the project, with randomized values for security tokens.

Syntax:

```
Usage: alg generate_settings [OPTIONS]

    Generate new environment variables for your algorithm project.

Options:
  -p, --production  Use production settings
  -h, --help        Show this message and exit.
```

This is particularly useful when deploying an algorithm project to production, so that you don't have to worry about transporting sensitive information to the new environment.

Using the `-p` option sets `FLASK_ENV` to "production", thus disabling debug mode for the project.

info

Get information about an algorithm installed in the Algorithm Registry.

Syntax:

```
Usage: alg info [OPTIONS] ALGORITHM

    Get detailed info about an algorithm from the Registry.

Options:
  -r, --registry TEXT  Get algorithm info from which registry?
  -v, --version TEXT    Specify an algorithm version
  -h, --help           Show this message and exit.
```

If you have access to a private Registry, you can use the `-r` option to query that registry.

Use the `-v` option if you want to specify which version of an algorithm you want information on.

install

Install an algorithm from the Algorithm Registry.

Syntax:

```
Usage: alg install [OPTIONS] ALGORITHM

    Install an algorithm from the Registry.

Options:
  -r, --registry TEXT  Install algorithm from which registry?
  -v, --version TEXT    Specify an algorithm version
  -h, --help           Show this message and exit.
```

If you have access to a private Registry, you can use the `-r` option to install from that registry.

Use the `-v` option if you want to specify which version of an algorithm you want to install.

list

List algorithms in the current project.

Syntax:

```
Usage: alg list [OPTIONS]

    List algorithms in the current project.

Options:
  -h, --help  Show this message and exit.
```

Algorithms will be displayed in a tidy tabular format:

```
>>> alg list

Algorithm Name      Version      Description
-----
↩-----
(continues on next page)
```


(continued from previous page)

multiply_numbers	0.0.1	Multiply two numbers together to get a result.
output_image_to_client	0.0.1	This algorithm will pull the path to an image (RGB ↪ ↪png currently supported)...
add_numbers	0.0.1	Add two numbers together to get a result.
getmaptiles_roi	0.0.1	This algorithm will gather up map tiles at a given ↪ ↪zoom level that intesect...
subtract_numbers	0.0.1	Subtract one number from another to get a result.
divide_numbers	0.0.1	Divide one number from another to get a result.
my_first_algorithm	0.0.1	This is my first algorithm, and I'm proud of it.
stitch_tiles	0.0.1	This algorithm stitches a group of map tiles saved ↪ ↪in a directory together....

publish

Publish an algorithm to the Algorithm Registry.

Syntax:

```
Usage: alg publish [OPTIONS] ALGORITHM_NAME

    Publish an algorithm to the Algorithm Registry.

Options:
  -r, --registry TEXT  Publish to which registry?
  -h, --help           Show this message and exit.
```

If you have access to a private Registry, you can use the `-r` option to publish the algorithm to that registry.

run

Run the ATK development environment.

Syntax:

```
Usage: alg run [OPTIONS] [ARGS]...

Options:
  --help  Show this message and exit.
```

If you're running on a virtual server (such as with VirtualBox), you can use the `--host=0.0.0.0` option.

search

Search the Algorithm Registry for an algorithm by name.

Syntax:

```
Usage: alg search [OPTIONS] SEARCH_STRING

    Search the Algorithm Registry for algorithm names matching a string.

Options:
  -r, --registry TEXT  Search algorithms from which registry?
  -h, --help           Show this message and exit.
```

If you have access to a private Registry, you can use the `-r` option to search that registry.

shell

Launch a Python shell with the current algorithm project settings active.

Syntax:

```
Usage: alg shell [OPTIONS] [ARGS]...

Options:
  --help  Show this message and exit.
```

test

Test one or more algorithms in the current project.

Syntax:

```
Usage: alg test [OPTIONS] [ALGORITHM]

Test algorithms in your project. You may specify an algorithm to test:
  >>> alg test my_algorithm

or test all algorithms:
  >>> alg test

Options:
  -h, --help  Show this message and exit.
```

uninstall

Remove an algorithm from the current project.

Syntax:

```
Usage: alg uninstall [OPTIONS] ALGORITHM

Remove an algorithm from your project.

Options:
  -h, --help  Show this message and exit.
```

1.12 API

This page documents the inheritable classes provided by the ATK, and their various attributes and public functions.

class Algorithm (`[cl=None[, params=None]]`)

Parameters

- **cl** (`ChainLedger`) – ChainLedger object instantiation
- **params** (`dict`) – Python dictionary containing values for algorithm inputs

- **errors** (*list*) – List of errors generated by an algorithm
- **logger** – Python logging object
- **name** (*string*) – Name of algorithm

run ()

Override this function with your algorithm logic.

raise_client_error (*err*)

Parameters **err** (*string*) – String containing the error message

Returns AlgorithmException object

Return type AlgorithmException

Raise an error to the calling application, such as a web browser.

class AlgorithmChain (*path, passed_chain*)

Parameters

- **atk_path** (*string*) – Folder location for the current algorithm project
- **chain_name** (*string*) – Name of the algorithm chain as it appears in the chain definition
- **chain_definition** (*dict*) – Python dictionary containing chain definition

get_request_dict ()

Return type dict

Return a dict containing all algorithms in the current chain and their default parameter values as if making a request to run a chain (e.g.: from a web form).

class ChainLedger (*status_key*)

Parameters

- **status_key** (*string*) – Unique ID for chain run, used for obtaining status and history
- **metadata** (*dict*) – key, value pairs placed on the Chain Ledger by algorithms
- **history** (*list*) – array of metadata, ordered by chain run order
- **chain_percent** (*int*) – percent progress of chain run
- **batch_percent** (*int*) – percent progress of batch run

add_to_metadata (*key, value*)

Parameters

- **key** (*string*) – metadata key used for later retrieval
- **value** (*any*) – value to be saved with the key

Add a key, value pair to the Chain Ledger

get_from_metadata (*key*)

Parameters **key** (*string*) – metadata key used to retrieve the item

Returns value of key in metadata

Return type any

Return a value placed in Chain Ledger metadata under *key*.

get_from_history (*history_index, key*)

Parameters

- **history_index** (*int*) – position in the Chain Ledger history for the algorithm sought
- **key** (*string*) – key to be located in metadata

Returns value in metadata history under key

Return type any

Search the Chain Ledger history for a key in metadata, based on the index of the algorithm in the history list.

search_history (*key*, *algorithm_name*)

Parameters

- **key** (*string*) – key to search for
- **algorithm_name** (*string*) – name of algorithm to find in history

Returns a set of items matching the algorithm, key pair

Return type list

Search the history for a particular metadata key coming from an algorithm, however many times it occurs.

is_algo_in_history (*algorithm_name*)

Parameters **algorithm_name** (*string*) – name of algorithm to search

Return type boolean

Return True if algorithm name appears in the history, otherwise return False.

set_status (*status* [, *percent=0*])

Parameters **status** (*string*) – status message for calling application

Set a key in app.config with the Chain Ledger status_key as its name, and create a dictionary of status information, setting the app.config[key] value equal to that dictionary. Example:

```
{
    "all_msg": "",
    "latest_msg": "",
    "algorithm_percent_complete": 0,
    "chain_percent_complete": 0,
    "batch_percent_complete": 0
}
```

get_results_folder ()

Returns location of results folder

Return type string

Provide the location of the results folder for files to be saved during an algorithm chain run

get_temp_folder ()

Returns location of temp folder

Return type string

Provide the location of the temp folder for files to be saved during an algorithm chain run. These files will be deleted at the end of the chain run.

clear_temp_folder ()

Delete all files and folders within the chain temp folder

class AlgorithmTestCase

Test Case template class to use for conducting unit tests of algorithms. Inherits from unittest.TestCase.

check_metadata (*key*, *value*)

Parameters

- **key** (*string*) – key in metadata to validate by
- **value** (*any*) – value to validate

Return type boolean

Return True if value is the same as that found on the metadata. Otherwise return False.

check_status (*status*)

Parameters **status** (*string*) – string to validate

Return type boolean

Return True if status is the same as that found on the app.config dictionary under the Chain Ledger status key. Otherwise return False.

1.13 Docker

1.13.1 Building the atk Docker image

If you have [Docker](#) installed, you can build the base Docker image for the ATK:

Note: If you are running Docker on Windows instead of Linux, make sure you have CPU Virtualization enabled in your motherboard BIOS

```
$ git clone https://github.com/BeamIO-Inc/algorithm_toolkit.git
$ cd algorithm_toolkit
$ docker build -t atk -f ./docker/atk/Dockerfile .
```

Visit [here](#) to learn more about base Docker images.

1.13.2 Running the atk Docker image (Optional)

In theory, a user could work purely with this Docker image, however, see [example-project](#) for its intended use.

The following command can then be used to run this Docker image:

```
$ docker run -d -p 5000:5000 --name atk-container atk
```

The container can then be accessed via:

```
$ docker exec -it atk-container /bin/bash
```

1.13.3 Building the atk-dev Docker image

This build requires that you have already built the atk Docker image in the previous steps.

With this requirement satisfied, you can proceed with the build:

```
$ docker build -t atk-dev -f ./docker/atk-dev/Dockerfile .
```

This Docker image contains the contents in the atk image, and the [JupyterLab](#) web based user interface.

1.13.4 Running the atk-dev Docker image

```
$ docker run -d -p 5000:5000 -p 8888:8888 atk-dev
```

JupyterLab allows you to open a terminal, which gives you full access to your container through your web browser (Pretty cool!).

1.13.5 Running the ATK from the JupyterLab terminal

Once you open up a terminal, you can go ahead and create a project:

```
$ alg cp myproject
```

To run this project there is one extra little trick.

You will need to specify the Docker container host IP, which will always be 0.0.0.0:

```
$ cd myproject
$ alg run --host 0.0.0.0
```

Now, you can access the ATK web interface by opening another web browser on your local machine and going to <http://localhost:5000/>.

1.13.6 Recommendations for atk-dev

Set an access token for JupyterLab

For security reasons, JupyterLab recommends that you set an access token.

You define the access token when running the atk-dev docker image:

```
$ docker run -d -e JUPYTER_TOKEN=enter_your_token_here -p 5000:5000 -p 8888:8888 atk-
↪dev
```

Go to <http://localhost:8888/> using your local web browser.

Enter the token in the ‘Password or token’ field, and click login.

After logging in, you can log out and set a password if you would like.

Note that the password will not go into effect until the Docker container is restarted.

Volume mounting

Earlier we created a project in our Docker container through the JupyterLab terminal.

But what if something happens to our container down the road (Whether it becomes corrupt, or we accidentally remove it)?

Well... all the work inside the container would be lost

Volume mounting allows our container to store our work on a directory on our local machine.

This way, if something happens to our container, we will maintain all of the work we have done.

- You can learn more about volume mounting [here](#).

Volume mounting is defined when running the atk-dev Docker image:

```
$ docker run -d -v /full_path_to_local_directory/:/opt/workspace -p 5000:5000 -p
↪8888:8888 atk-dev
```

`/full_path_to_local_directory/` represents the full path to a directory on your local machine.

`/opt/workspace` will be a new directory in the Docker container, which will store our work on our local machine, in `/full_path_to_local_directory/`.

1.14 The Algorithm Registry

The [Algorithm Registry](#) is a repository of algorithms developed by the community of ATK users. If you make algorithms, we hope that you will consider publishing them to the Registry.

1.14.1 Searching for Algorithms

Your primary interface to the Algorithm Registry as an ATK user is the [CLI](#).

Finding algorithms to use in your project is easy:

```
$ alg search name_to_search_for
```

You can also use part of a name:

```
$ alg search pdal
```

A list of algorithms that match your search string will appear in your terminal window.

1.14.2 Getting Algorithm Info

Once you know the name of your algorithm, you can get details about it:

```
$ alg info getmaptiles_roi
```

1.14.3 Installing Algorithms

When you want to install an algorithm into your project, do it like so:

```
$ alg install getmaptiles_roi
```

1.14.4 Publishing Algorithms

When you've created your awesome algorithm and you want to share it with others, we make it pretty easy to do so. You will need to do a couple of things first:

Get a TileDriver account

To publish algorithms, you need a TileDriver account. [You can get one for free.](#)

Once you have an account and have logged in, go to the [Algorithm Toolkit page](#) for instructions on how to link your account with the Registry.

Algorithm Naming

Every algorithm in the Registry has a unique name. You need to check the Registry before publishing to ensure your algorithm's name is not already taken.

Namespacing

One helpful way to ensure your algorithm's name will be unique is to use **namespacing**. This is simple to do: just add “*namespace*” to the beginning of your algorithm's name.

For example, if you chose the namespace “crunchy_frog”, you would create algorithms with names like “crunchy_frog/my_first_algorithm”.

Note: Namespaces must only contain letters, numbers or the underscore (“_”). By convention namespaces are all lowercase, but that is not required.

Namespaces must end with a single forward slash (“/”). No other slashes may be in the namespace or name of the algorithm.

Make sure your algorithm is complete

We ask that everyone who publishes algorithms ensure that they contain the following:

- The algorithm's folder **MUST** contain at least a `main.py` and `algorithm.json` file.
- A version string (see note below).
- A full description of what the algorithm does, including any dependencies users will need to install.
- Each parameter and output in your algorithm should also have a clear description and help text.
- The algorithm should have a good README file in the algorithm folder.
- A LICENSE file should also be in the algorithm folder providing clear terms of use.
- Unit tests for the algorithm are very helpful; follow the instructions in [Tutorial: Do Maths](#) for writing and running algorithm tests.

The good news is: if you use the development environment, all of these things should already be in your algorithm. See [Creating Your First Algorithm](#) for details on how to create algorithms properly for publishing.

Another helpful but not required file is one called CHANGELOG. If you create one of these, each version of the algorithm you publish can contain information about changes since the prior version. The CHANGELOG should use [Markdown syntax](#).

Publish!

Once you've done these things, go ahead and publish your algorithm:

```
$ alg publish my_algorithm
```

This will zip up the contents of your algorithm's folder, including all scripts and files it uses within that folder, and add it to the Registry. A copy of the zip file will be created in a folder called **algorithm_uploads** in your algorithm project.

Note: The size of the zip file cannot be larger than 25 MB. If it is, the ATK will refuse to publish it.

A note about versions

The version string in your `algorithm.json` file is important. The ATK scans this string and compares it with what has already been published on the Registry. Importantly, it does not try to guess whether this version is older or newer than what is in the Registry: it assumes that whatever version you are uploading is the current one.

If the version number is the same as one already in the Registry, what you upload will overwrite that version AND THAT VERSION WILL BECOME CURRENT. Therefore, if you want to make adjustments to a prior version of an algorithm, you may have to publish twice in order to ensure that the version you wish to be current is current.

Obviously, before you publish an algorithm make sure you have updated the version number.

Users can also view or install specific versions of an algorithm:

```
$ alg info -v 0.1.2 your_algorithm
$ alg install -v 0.1.2 your_algorithm
```

If not specified, the current version will be used in these commands. An equivalent command would be:

```
$ alg info -v current your_algorithm
```

Note that you do not use the `-v` option when publishing.

Note: You can use any versioning scheme you wish in your algorithms. Instead of a number, you could use “Beta”, “v2”, “Fred” or anything else you like.

1.14.5 About Licensing Your Algorithm

As you’ve probably guessed, we place a lot of emphasis on algorithms having some version of an [Open Source license](#). However, you can put whatever restrictions you wish on usage of your algorithm.

We provide several Open Source license types out of the box with the ATK:

- MIT
- BSD-3-Clause
- GNU AGPLv3
- GNU GPLv3
- GNU LGPLv3
- Mozilla
- Apache
- The Unlicense

If you select one of these when creating your algorithm, the corresponding license text will be added to the algorithm’s `LICENSE` file.

We also provide two other license options:

- Proprietary

- See LICENSE File

Both are essentially equivalent when creating your algorithm: the LICENSE file will be blank, and you can paste in any terms you wish (such as another Open Source license if we don't provide it).

The difference between these options is that choosing "Proprietary" is a signal to users that you require them to contact you and negotiate rights to use the algorithm. In any case, it is important that you place the terms of use in the LICENSE file.

License Enforcement

As stated in the Registry Terms of Use, we do not attempt to enforce your license terms. Publishing your algorithm to the Registry is done at your own risk.

1.15 Contributing to the ATK

Thanks for your interest in contributing to the Algorithm Toolkit codebase! You should know a few things.

1.15.1 Code of Conduct

First of all: this project has a code of conduct. Please read the CODE_OF_CONDUCT file in the project root folder and stick to its principles.

1.15.2 License

The MIT license (see LICENSE file) applies to all contributions.

1.15.3 Issue Conventions

We use GitHub's Issue functionality to track issues submitted by users and developers. We ask that you abide by a few rules:

If you have questions about installation, distribution, and usage of the ATK, please first review the project's fairly decent documentation. If you open an issue of this kind, you will be reminded of the importance of reading the documentation. Indicate you have done so in your issue.

Questions about development of the ATK, brainstorming, requests for comment, and not-yet-actionable proposals should be prefaced with the word IDEA: at the top of the issue, on its own line. Please understand that although we enjoy getting these we may not respond at all to the issue once it's posted. We may also close the issue without responding, which generally means we've either logged it in our internal tracking system or we're not going to do anything with it at all. We will try to indicate which in a response.

Please search existing issues, open and closed, before creating a new one.

The following details would be helpful to know when you post bug reports, so please include them:

- Operating system type and version (Windows? Ubuntu 16.04? 18.04?)
- The version and source of the ATK (PyPI, GitHub, or somewhere else?)
- Any external dependencies your project has (particularly gnarly ones to troubleshoot like GDAL are good to know about)

Please provide these details as well as tracebacks and relevant logs.

1.15.4 Design Principles

If there is a spectrum between a “batteries included” and “batteries not included” open source project (and we think there is), the ATK tends to want to be a “batteries included” project. That said, we want to make the ATK easy to install and use; therefore, contributions that increase the burden on the end user of the Toolkit are to be avoided.

Multiple interfaces

You’ve probably noticed from reading these docs that we provide multiple paths to using the ATK. The two principal means are through a Web browser interface and a command line interface (CLI). Please consider how your new or revised feature would be used by both of these interfaces.

We believe that providing a fully featured Web interface makes a big difference to the usability of this project. Make sure that your contributions take that into account, and enhance the usability of the ATK as well as add functionality.

Scientific computing roots

The developers of this project are deeply rooted in the scientific community, particularly in the realm of Physics known as Imaging Science. As a result, it is common to see ATK-based projects depending on packages that are geared toward science and mathematics (Python packages like NumPy, SciPy, Matplotlib and so on) as well as C-based programs in the imaging and geospatial areas (particularly GDAL, OpenCV and proj.4).

While true, this does not mean that we are only interested in contributions from this community. We believe the ATK has broad appeal for any type of data processing, and we hope that we have a diverse set of contributor backgrounds.

1.15.5 Code Conventions

The ATK is first and foremost a Python project.

The ATK supports Python 2.7 and Python 3. It will likely continue to support Python 2 beyond the official end of life of that version ([currently January 1, 2020](#)). Contributions are expected to support both major versions.

We strongly prefer code adhering to [PEP8](#).

Tests are mandatory for new features. We use Python’s [unittest](#). There is a `.coveragerc` file in the root folder for configuring the [coverage utility](#), which is a handy tool for ensuring code coverage.

We aspire to 100% test coverage for the ATK, but as of this writing we are not there yet. We welcome any contributions that enhance our code coverage for testing. In particular, we do not have any tests for JavaScript code nor any end-to-end tests.

1.15.6 Development Environment

Developing the ATK requires Python 2.7 or any final release after and including 3.6. See above for version adherence guidelines.

Initial Setup

First, clone the ATK’s `git` repo:

```
$ git clone https://github.com/BeamIO-Inc/algorithm_toolkit
```

Development should occur within a [virtual environment](#) to better isolate development work from custom environments. For Python 2 environments, we recommend either [Miniconda](#) or [virtualenvwrapper](#).

All your work should be done in a separate branch from master. Currently we use the [Feature Branch Workflow](#), and we ask that you do the same. Once you feel it's ready (including having unit tests), you may make a pull request on GitHub.

Running the tests

The project's tests currently live in a single file called `test_main.py` in the project root folder. This will likely change in future.

To run the entire suite and the code coverage report:

```
$ pip install coverage
$ coverage run -m unittest discover
$ coverage html
```

A single test:

```
$ coverage run -m unittest test_main.NAME_OF_TEST_CLASS.name_of_test
$ coverage html
```

You can also run without coverage:

```
$ python -m unittest discover
```

or:

```
$ python -m unittest test_main.NAME_OF_TEST_CLASS.name_of_test
```

Note that many of the tests create and destroy a test ATK project within the ATK folder structure. Your `setUp` and `tearDown` methods should follow the example set by the `ATKTestCase` class.

Place shared test utilities and mocked data in `t_utils.py`.

1.15.7 Contributing to Docs

We also welcome contributions to these docs. You should follow the same workflow above for contributing code as for contributing documentation. Also, please follow the `reStructuredText` format used by the existing documents ([guidelines here](#)).

You will need to install [Sphinx](#) and the [RTD theme](#) to build docs locally (which you should do to make sure they look OK).

```
$ pip install sphinx sphinx_rtd_theme
$ sphinx-build docs docs/html -a
```

1.16 Indices and tables

- [genindex](#)
- [modindex](#)
- [search](#)

A

Algorithm (*built-in class*), 70
 Algorithm.raise_client_error() (*built-in function*), 71
 Algorithm.run() (*built-in function*), 71
 AlgorithmChain (*built-in class*), 71
 AlgorithmChain.ChainLedger (*built-in class*), 71
 AlgorithmChain.ChainLedger.add_to_metadata() (*built-in function*), 71
 AlgorithmChain.ChainLedger.clear_temp_folder() (*built-in function*), 72
 AlgorithmChain.ChainLedger.get_from_history() (*built-in function*), 71
 AlgorithmChain.ChainLedger.get_from_metadata() (*built-in function*), 71
 AlgorithmChain.ChainLedger.get_results_folder() (*built-in function*), 72
 AlgorithmChain.ChainLedger.get_temp_folder() (*built-in function*), 72
 AlgorithmChain.ChainLedger.is_algo_in_history() (*built-in function*), 72
 AlgorithmChain.ChainLedger.search_history() (*built-in function*), 72
 AlgorithmChain.ChainLedger.set_status() (*built-in function*), 72
 AlgorithmChain.get_request_dict() (*built-in function*), 71
 AlgorithmTestCase (*built-in class*), 72
 AlgorithmTestCase.check_metadata() (*built-in function*), 72
 AlgorithmTestCase.check_status() (*built-in function*), 72
 ATK_API_KEY (*built-in variable*), 12
 ATK_MANAGEMENT_API_KEY (*built-in variable*), 12

B

binary (*built-in variable*), 48

C

CORS_ORIGIN_WHITELIST (*built-in variable*), 11
 csv (*built-in variable*), 47

D

DEFAULT_WORKING_ROOT (*built-in variable*), 11

E

evenonly (*built-in variable*), 22

F

FLASK_ENV (*built-in variable*), 12
 FLASK_SECRET_KEY (*built-in variable*), 11

G

georaster (*built-in variable*), 47
 geojson (*built-in variable*), 47
 greaterthan (*built-in variable*), 21

J

json (*built-in variable*), 47

L

lessthan (*built-in variable*), 21

O

oddonly (*built-in variable*), 22

T

text (*built-in variable*), 48